> Эта страница была переведена с английского языка силами сообщества. Вы тоже можете внести свой вклад, присоединившись к русскоязычному сообществу MDN Web Docs.

Регулярные выражения

Регулярные выражения - это шаблоны, используемые для сопоставления последовательностей символов в строках. В JavaScript регулярные выражения также являются объектами. Эти шаблоны используются в методах `exec` (en-US) и `test` (en-US) объекта `RegExp` (en-US) а также `match` (en-US), `replace`, `search` (en-US), `split` (en-US) объекта `String`. Данная глава описывает регулярные выражения в JavaScript.

Создание регулярного выражения

Регулярное выражение можно создать двумя способами:

- Используя литерал регулярного выражения, например:

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

  Литералы регулярных выражений вызывают предварительную компиляцию регулярного выражения при анализе скрипта. Если ваше регулярное выражение постоянно, то пользуйтесь им, чтобы увеличить производительность.

- Вызывая функцию конструктор объекта `RegExp` (en-US), например:

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

  Использование конструктора влечёт за собой компиляцию регулярного выражения во время исполнения скрипта. Используйте данный способ, если знаете, что выражение будет изменяться или не знаете шаблон заранее. Например вы получаете его из стороннего источника, при пользовательском вводе.

Написание шаблона регулярного выражения

Шаблон регулярного выражения состоит из обычных символов, например `/abc/`, или комбинаций обычных и специальных символов, например `/ab*c/` или `/Chapter (\d+)\.\d*/`. Последний пример включает в себя скобки, которые используются как "запоминающий механизм". Соответствие этой части шаблона запоминается для дальнейшего использования, как описано в Использование совпадений подстрок заключённых в скобки.

Использование простых шаблонов

Простые шаблоны используются для нахождения прямого соответствия в тексте. Например, шаблон `/abc/` соответствует комбинации символов в строке только когда символы 'abc' встречаются вместе и в том же порядке. Такое сопоставление произойдёт в строке "Hi, do you know your abc's?" и "The latest airplane designs evolved from slabcraft." В обоих случаях сопоставление произойдёт с подстрокой 'abc'. Сопоставление не произойдёт в строке "Grab crab", потому что она не содержит подстроку 'abc'.

Использование специальных символов

В случае когда поиск соответствия требует чего-то большего, чем прямое сопоставление, например нахождение последовательности символов 'b' или нахождение пробела, шаблон включает в себя специальные символы. Например, шаблон `/ab*c/` соответствует любой комбинации символов, в которой за 'a' следует ноль или более символов 'b' (`*` означает ноль или

более вхождений предыдущего символа), за которыми сразу же следует символ 'c'. В строке "cbbabbbbcdebc," этому шаблону сопоставляется подстрока 'abbbbc'.

В следующей таблице приводится полный список специальных символов регулярных выражений с их описаниями.

Таблица 4.1 Специальные символы в регулярных выражения.

Символ	Значение
\	Одно из следующего: • Для символов обычно обрабатываемых буквально, означает что следующий символ является специальным и не должен интерпрети буквально. Например, /b/ сопоставляется символу 'b'. Добавляя слеш перед b, т.е используя /\b/ , символ становится специальным си означающим границу слова. • Для символов обычно обрабатываемых особым образом означает, что следующий символ не является специальным и должен интер буквально. Например, * является специальным символом, сопоставляемым 0 или более повторений предыдущего символа; например, /a*/ соответствие 0 или более символов a. Для буквальной интерпретации *, поставьте перед ней обратный слеш; например, /a* 'a*'. Также не забудьте заэкранировать сам \ при его использовании в записи new RegExp("pattern") поскольку \ также является символом в обычных строках.
^	Соответствует началу ввода. Если установлен флаг многострочности, также производит сопоставление непосредствен переноса строки. Например, /^A/ не соответствует 'A' в "an A", но соответствует 'A' в "An E". Этот символ имеет другое значение при появлении в начале шаблона набора символов. Например, /[^a-z\s]/ соответствует 'I' в "I have 3 sisters".
$	Соответствует концу ввода. Если установлен битовый флаг многострочности, также сопоставляется содержимому до г Например, /t$/ не соответствует 't' в строке "eater", но соответствует строке "eat".
*	Соответствует предыдущему символу повторенному 0 или более раз. Эквивалентно {0,}. Например, /bo*/ соответствует 'boooo' в "A ghost booooed" и 'b' в "A bird warbled", но не в "A goat grunted".
+	Соответствует предыдущему символу повторенному 1 или более раз. Эквивалентно {1,}. Например, /a+/ соответствует 'a' в "candy" и всем символам 'a' в "caaaaaaandy".
?	Соответствует предыдущему символу повторенному 0 или 1 раз. Эквивалентно {0,1}.

Символ	Значение
	Например, `/e?le?/` соответствует 'el' в "angel" и 'le' в "angle" а также 'l' в "oslo". Если использован сразу после квалификаторов `*`, `+`, `?`, или `{}`, делает квалификатор "нежадным" (соответствующим м количеству символов), в отличие от режима по умолчанию, являющимся "жадным" (соответствующим максимальному чи Например, используя /\d+/ не глобальное сопоставление "123abc" возвращает "123", если использовать /\d+?/, только "1" возвращена. Также используется в упреждающих утверждениях (assertions), описанных в строках x(?=y) и x(?!y) данной таблицы.
.	(десятичная точка) соответствует любому символу кроме переноса строки. Например, `/.n/` соответствует 'an' и 'on' в "nay, an apple is on the tree", но не 'nay'.
(x)	Соответствует 'x' и запоминает это соответствие. Это называется захватывающие скобки. Например, `/(foo)/` соответствует 'foo' в "foo bar." Сопоставленная строка может быть получена из элементов результиру `[1]`, ..., `[n]`.
(?:x)	Соответствует 'x' но не запоминает соответствие. Это называется не-захватывающие скобки. Сопоставленная строка не может б элементов результирующего массива `[1]`, ..., `[n]`.
x(?=y)	Соответствует 'x' только если за 'x' следует 'y'. Это называется упреждение. Например, `/Jack(?=Sprat)/` соответствует 'Jack' только если за ним следует 'Sprat'. `/Jack(?=Sprat|Frost)/` соответствует если за ним следует 'Sprat' или 'Frost'. Тем не менее, ни 'Sprat' ни 'Frost' не являются частью сопоставленного результата.
x(?!y)	Соответствует 'x' только если за 'x' не следует 'y'. Это называется отрицательное упреждение. Например, `/\d+(?!\.)/` соответствует числу только если за ним не следует десятичная точка. Регулярное выражение `/\(?!\.)/.exec("3.141")` сопоставит '141' но не '3.141'.
x|y	Соответствует либо 'x' либо 'y'. Например, `/green|red/` соответствует 'green' в "green apple" и 'red' в "red apple."
{n}	n - положительное целое. Соответствует ровно `n` вхождениям предыдущего символа. Например, `/a{2}/` не соответствует 'a' в "candy," но соответствует всем a в "caandy," первым двум a в "caaandy."
{n,m}	m и n - положительные целые. Соответствует как минимум `n` и максимум `m` вхождениям предыдущего символа. При m= пропускается. Например, `/a{1,3}/` ничему не соответствует в строке "cndy", символу 'a' в "candy," двум a в "caandy," и трём первым a в Отметим, что при сопоставлении "caaaaaaandy", совпадает "aaa", хотя изначальная строка содержит больше a.

Символ	Значение
`[xyz]`	Набор символов. Соответствует любому символу из перечисленных. Можно указать диапазон символов, используя тире символы (как точка (`.`) и звёздочка (`*`)) не имеют специального значения внутри такого набора. Их не надо экранироват работает также. Например, `[abcd]` эквивалентна `[a-d]`. Они соответствуют 'b' в "brisket" и 'c' в "city". `/[a-z.]+/` и `/[\w.]+/` обе соответст "test.i.ng".
`[^xyz]`	Инвертированный или дополняющий набор символов. Это означает соответствие всему, что не в скобках. Можно указать символов с помощью тире. Все, что действует в обычном наборе символов, действует и здесь. Например, `[^abc]` эквивалентно `[^a-c]`. Они соответствуют изначально 'r' в "brisket" и 'h' в "chop."
`[\b]`	Соответствует бэкспейсу (U+0008). (Не путать с `\b`.)
`\b`	Соответствует границе слова. Граница слова соответствует позиции, где за символом слова не следует другой символ предшествует ему. Отметим, что граница слова не включается в соответствие. Другими словами, длина сопоставленной равна нулю. (Не путать с `[\b]`.) Примеры: `/\bmoo/` соответствует 'moo' в слове "moon" ; `/oo\b/` не соответствует 'oo' в слове "moon", поскольку за 'oo' следует символ 'n' , являющимся символом слова; `/oon\b/` соответствует 'oon' в слове "moon", поскольку 'oon' является окончанием строки, и таким образом, за этими сим следует другой символ слова; `/\w\b\w/` никогда не будет ничему соответствовать, поскольку за символом слова никогда не может следовать и границ символ слова. `Note:` JavaScript's regular expression engine defines a `specific set of characters` to be "word" characters. Any character is considered a word break. This set of characters is fairly limited: it consists solely of the Roman alphabet in both upper- a decimal digits, and the underscore character. Accented characters, such as "é" or "ü" are, unfortunately, treated as word bre
`\B`	Соответствует несловообразующей границе. Несловообразующая граница соответствует позиции, в которой предыдущ символы являются символами одного типа: либо оба должны быть словообразующими символами, либо несловообразу конец строки считаются несловообразующими символами. Например, `/\B../` соответствует 'oo' в слове "noonday" (, а `/y\B./` соответствует 'ye' в "possibly yesterday."
`\c`*`X`*	Где *X* является символом случайного выбора из последовательности от A до Я. Соответствует управляющему символу Например, `/\cM/` соответствует control-M (U+000D) в строке.
`\d`	Соответствует цифровому символу. Эквивалентно выражению `[0-9]`. Например, `/\d/` or `/[0-9]/` соответствует '2' в "B2 is the suite number."

Символ	Значение
`\D`	Соответствует любому нецифровому символу. Эквивалентно выражению `[^0-9]`. Например, `/\D/` or `/[^0-9]/` соответствует 'B' в предложении "B2 is the suite number."
`\f`	Соответствует символу прогона страницы (U+000C). Особый символ управления печатью.
`\n`	Соответствует символу перевода строки (U+000A).
`\r`	Соответствует символу возврата каретки (U+000D).
`\s`	Соответствует одиночному символу пустого пространства, включая пробел, табуляция, прогон страницы, перевод строк `[ \f\n\r\t\v\u00A0\u1680\u180e\u2000\u2001\u2002\u2003\u2004\u2005\u2006\u2007\u2008\u2009\u200a\u2028\u2029\u2028\u2029\u2` Например, `/\s\w*/` совпадает с ' bar' в "foo bar."
`\S`	Соответствует одиночному символу непустого пространства. Эквивалентен `[^ \f\n\r\t\v\u00A0\u1680\u180e\u2000-\u200a\u2028\u2029\u202f\u205f\u3000]`. Например, `/\S\w*/` совпадает с 'foo' в "foo bar."
`\t`	Соответствует символу горизонтальной табуляции (U+0009).
`\v`	Соответствует символу вертикальной табуляции (U+000B).
`\w`	Соответствует любому цифробуквенному символу включая нижнее подчёркивание. Эквивалентен `[A-Za-z0-9_]`. Например, `/\w/` совпадает с 'a' в "apple," '5' в "$5.28," и '3' в "3D."
`\W`	Соответствует любому не цифробуквенному символу. Равносилен `[^A-Za-z0-9_]`. Например, `/\W/` or `/[^A-Za-z0-9_]/` совпадает с '%' в "50%."
`\n`	Где *n* это положительное целое, обратная ссылка на последнюю найденную подстроку, соответствующую *n*, заключённ скобки в регулярном выражении (считая левые скобки). Например, `/apple(,)\sorange\1/` соответствует 'apple, orange,' в "apple, orange, cherry, peach."
`\0`	Соответствует символу NULL (U+0000). Не следует ставить за ним другой цифровой символ, поскольку `\0<digits>` является вос экранирующей последовательностью.
`\xhh`	Соответствует символам кода hh (две шестнадцатеричные цифры)
`\uhhhh`	Соответствует символам кода hhhh (четыре шестнадцатеричные цифры).

Экранирование пользовательского ввода, соответствующего буквенной строке внутри регулярного выражения, может быть достигнуто простой заменой:

```
JS
function escapeRegExp(string) {
  return string.replace(/[.*+?^${}()|[\]\\]/g, "\\$&"); // $& means the whole matched string
}
```

Использование скобок

Скобки вокруг любой части регулярного выражения означают что эта часть совпадаемой подстроки будет запомнена. Раз запомнена, подстрока может выбрана для использования как это описано в Using Parenthesized Substring Matches.

Например, паттерн `/Chapter (\d+)\.\d*/` включает в себя дополнительные экранирующие и специальные символы и указывает на то, что часть шаблона должна быть запомнена. Он точно соответствует символам слова 'Chapter ', за которыми следует один или более цифровых символов (`\d` означает любой цифровой символ, а `'+'` означает 1 или более раз), за которым следует десятичная точка (сама по себе являющаяся специальным символом; предшествующий десятичной точке слеш '\' означает, что паттерн должен искать литеральный символ '.'), после которой следует любой цифровой символ 0 или более раз (`'\d'` обозначает цифровой символ, `'*'` обозначает 0 или более раз). Кроме того, круглые скобки используются для запоминания первых же совпавших цифровых символов.

Этот шаблон будет найден во фразе "Open Chapter 4.3, paragraph 6" и цифра '4' будет запомнена. Но он не будет найден во фразе "Chapter 3 and 4", поскольку эта строка не имеет точки после цифры '3'.

Для того, чтобы сопоставить подстроку без вызова совпавшей части для запоминания, внутри круглых скобок необходимо предварить паттерн сочетанием символов `' ?:'` . Например, шаблон `(?:\d+)` будет соответствовать одному или более цифровому символу, но не запомнит совпавших символов.

Работа с Регулярными Выражениями

Регулярные выражения используются в методах `test` и `exec` объекта `RegExp` и с методами `match` , `replace` , `search` , и `split` объекта `String` . Эти методы подробно объясняются в Справочнике JavaScript

Метод	Описание
exec (en-US)	Метод `RegExp`, который выполняет поиск совпадения в строке. Он возвращает массив данных.
test (en-US)	Метод `RegExp` , который тестирует совпадение в строке. Возвращает либо истину либо ложь.
match (en-US)	Метод `String` , который выполняет поиск совпадения в строке. Он возвращает массив данных либо `null` если совпадения отсутствуют.
search (en-US)	Метод `String`, который тестирует на совпадение в строке. Он возвращает индекс совпадения, или -1 если совпадений не будет найдено.
replace	Метод `String` , который выполняет поиск совпадения в строке, и заменяет совпавшую подстроку другой подстрокой, переданной как аргумент в этот метод.
split (en-US)	Метод `String`, который использует регулярное выражение или фиксированную строку чтобы разбить строку на массив подстрок.

Чтобы просто узнать есть ли в строке что либо соответствующее шаблону, воспользуйтесь методами `test` или `search` ; а чтобы получить больше информации пользуйтесь методами `exec` или `match` (хотя эти методы работают медленнее). Если вы пользуетесь `exec` или `match` и если совпадения есть, эти методы вернут массив и обновлённые свойства объекта ассоциированного регулярного выражения а также предопределённого объекта `RegExp` регулярного выражения. Если совпадений нет, метод `exec` вернёт `null` (который сконвертируется в `false`).

В след. примере, скрипт использует метод `exec` чтобы найти совпадения в строке.

```
JS
var myRe = /d(b+)d/g;
var myArray = myRe.exec("cdbbdbsbz");
```

Если вам не нужен доступ к свойствам регулярного выражения, то альтернативный способ получить `myArray` можно так:

```
JS
var myArray = /d(b+)d/g.exec("cdbbdbsbz");
```

Если вы хотите сконструировать регулярное выражение из строки, другой способ сделать это приведён ниже:

```
JS
var myRe = new RegExp("d(b+)d", "g");
var myArray = myRe.exec("cdbbdbsbz");
```

С помощью этих скриптов, поиск совпадения завершается и возвращает массив и обновлённые свойства показанные в след. таблице.

Table 4.3 Результаты выполнения регулярного выражения

Объект	Свойство или индекс	Описание	В этом примере.
myArray		Совпавшая строка и все запомненные подстроки.	["dbbd", "bb"]
	index	Индекс совпавшей подстроки (индекс начинается с нуля).	1
	input	Исходная строка.	"cdbbdbsbz"
	[0]	Последние совпавшие символы.	"dbbd"
myRe	lastIndex	Индекс с которого начнётся след. поиск совпадения. (Это свойство определяется только если регулярное выражение использует параметр g, описанный в [Advanced Searching With Flags](#Advanced_Searching_With_Flags).)	5
	source	Текст шаблона. Обновляется в момент создания регулярного выражения, а не во время выполнения.	"d(b+)d"

Как показано во втором варианте этого примера, вы можете использовать регулярное выражение, созданное при помощи инициализатора объекта, без присваивания его переменной. Таким образом, если вы используете данную форму записи без присваивания переменной, то в процессе дальнейшего использования вы не можете получить доступ к свойствам данного регулярного выражения. Например, у вас есть следующий скрипт:

```
JS
var myRe = /d(b+)d/g;
var myArray = myRe.exec("cdbbdbsbz");
console.log("The value of lastIndex is " + myRe.lastIndex);
```

Этот скрипт выведет:

```
The value of lastIndex is 5
```

Однако, если у вас есть следующий скрипт:

```
JS
```

```
var myArray = /d(b+)d/g.exec("cdbbdbsbz");
console.log("The value of lastIndex is " + /d(b+)d/g.lastIndex);
```

Он выведет:

```
The value of lastIndex is 0
```

Совпадения `/d(b+)d/g` в двух случаях являются разными объектами регулярного выражения и, следовательно, имеют различные значения для свойства `lastIndex`. Если вам необходим доступ к свойствам объекта, созданного при помощи инициализатора, то вы должны сначала присвоить его переменной.

Использование скобочных выражений для нахождения подстрок

Использование скобок в шаблоне регулярного выражения повлечёт "запоминание" совпавшей подстроки. Для примера, `/a(b)c/` вызовет совпадение 'abc' и запомнит 'b'. Чтобы получить совпадения скобочного выражения используйте `Array` elements `[1]`, ..., `[n]`.

Число возможных скобочных подстрок неограничено. Возвращаемый массив содержит все полученные совпадения, удовлетворяющие выражению в скобках. Следующий пример показывает как использовать скобочные выражения для нахождения подстрок.

Следующий скрипт использует метод `replace()`, чтобы поменять местами слова (символы) в строке. Для замены текста скрипт использует обозначения `$1` и `$2` для обозначения первого и второго совпадения скобочного выражения.

JS

```
var re = /(\w+)\s(\w+)/;
var str = "John Smith";
var newstr = str.replace(re, "$2, $1");
console.log(newstr);
```

Выведет "Smith, John".

Расширенный поиск с флагами

Регулярные выражения имеют четыре опциональных флага, которые делают возможным глобальный и регистронезависимый поиск. Флаги могут использоваться самостоятельно или вместе в любом порядке, а также могут являться частью регулярного выражения.

Flag	Description
g	Глобальный поиск.
i	Регистронезависимый поиск.
m	Многострочный поиск.
y	Выполняет поиск начиная с символа, который находится на позиции свойства lastindex текущего регулярного выражения.

Чтобы использовать флаги в шаблоне регулярного выражения используйте следующий синтаксис:

JS

```
var re = /pattern/flags;
```

или

JS

```
var re = new RegExp("pattern", "flags");
```

Обратите внимание, что флаги являются неотъемлемой частью регулярного выражения. Флаги не могут быть добавлены или удалены позднее.

Для примера, `re = /\w+\s/g` создаёт регулярное выражение, которое ищет один или более символов, после которых следует пробел и ищет данное совпадение на протяжении всей строки.

JS

```
var re = /\w+\s/g;
var str = "fee fi fo fum";
var myArray = str.match(re);
console.log(myArray);
```

Выведет ["fee ", "fi ", "fo "]. В этом примере вы бы могли заменить строку:

JS

```
var re = /\w+\s/g;
```

на следующую:

JS

```
var re = new RegExp("\\w+\\s", "g");
```

и получить тот же результат.

Флаг `m` используется, чтобы входная строка рассматривалась как многострочная. Если флаг `m` используется, то `^` и `$` вызовет совпадение в начале или конце любой строки в строке ввода вместо начала или конца вводимой строки целиком.

Примеры

След. примеры показывают использование регулярных выражений.

Изменение порядка в Исходной Строке

След. пример иллюстрирует формирование регулярного выражения и использование `string.split()` и `string.replace()`. Он очищает неправильно сформатированную исходную строку, которая содержит имена в неправильном порядке (имя идёт первым) разделённые пробелами, табуляцией и одной точкой с запятой. В конце, изменяется порядок следования имён (фамилия станет первой) и сортируется список.

JS

```
// The name string contains multiple spaces and tabs,
// and may have multiple spaces between first and last names.
var names = "Harry Trump ;Fred Barney; Helen Rigby ; Bill Abel ; Chris Hand ";

var output = ["---------- Original String\n", names + "\n"];

// Prepare two regular expression patterns and array storage.
// Split the string into array elements.

// pattern: possible white space then semicolon then possible white space
var pattern = /\s*;\s*/;

// Break the string into pieces separated by the pattern above and
// store the pieces in an array called nameList
var nameList = names.split(pattern);
```

```
// new pattern: one or more characters then spaces then characters.
// Use parentheses to "memorize" portions of the pattern.
// The memorized portions are referred to later.
pattern = /(\w+)\s+(\w+)/;

// New array for holding names being processed.
var bySurnameList = [];

// Display the name array and populate the new array
// with comma-separated names, last first.
//
// The replace method removes anything matching the pattern
// and replaces it with the memorized string—second memorized portion
// followed by comma space followed by first memorized portion.
//
// The variables $1 and $2 refer to the portions
// memorized while matching the pattern.

output.push("---------- After Split by Regular Expression");

var i, len;
for (i = 0, len = nameList.length; i < len; i++) {
  output.push(nameList[i]);
  bySurnameList[i] = nameList[i].replace(pattern, "$2, $1");
}

// Display the new array.
output.push("---------- Names Reversed");
for (i = 0, len = bySurnameList.length; i < len; i++) {
  output.push(bySurnameList[i]);
}

// Sort by last name, then display the sorted array.
bySurnameList.sort();
output.push("---------- Sorted");
for (i = 0, len = bySurnameList.length; i < len; i++) {
  output.push(bySurnameList[i]);
}

output.push("---------- End");

console.log(output.join("\n"));
```

Использование спецсимволов для проверки входных данных

В след. примере, ожидается что пользователь введёт телефонный номер и требуется проверить правильность символов набранных пользователем. Когда пользователь нажмёт кнопку "Check", скрипт проверит правильность введённого номера. Если номер правильный (совпадает с символами определёнными в регулярном выражении), то скрипт покажет сообщение благодарности для пользователя и подтвердит номер. Если нет, то скрипт проинформирует пользователя, что телефонный номер неправильный.

Внутри незахватывающих скобок `(?:` , регуляное выражение ищет три цифры `\d{3}` ИЛИ `|` открывающую скобку `\(` , затем три цифры `\d{3}` , затем закрывающую скобку `\)` , (закрывающая незахватывающая скобка `)`), затем тире, слеш, или десятичная точка, и когда это выражение найдено, запоминает символ `([-\/\.])` , следующие за ним и запомненные три цифры `\d{3}` , следующее соответствие тире, слеша или десятичной точки `\1` , и следующие четыре цифры `\d{4}` .

Регулярное выражение ищет сначала 0 или одну открывающую скобку `\(?` , затем три цифры `\d{3}` , затем 0 или одну закрывающую скобку `\)?` , потом одно тире, слеш или точка и когда найдёт это, запомнит символ `([-\/\.])` , след. три цифры `\d{3}` , followed by the remembered match of a dash, forward slash, or decimal point `\1` , followed by four digits `\d{4}` .

Событие "Изменить" активируется, когда пользователь подтвердит ввод значения регулярного выражения, нажав клавишу "Enter".

HTML

```
<!doctype html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" />
    <meta http-equiv="Content-Script-Type" content="text/javascript" />
    <script type="text/javascript">
     var re = /\(?\d{3}\)?([-\/\.])\d{3}\1\d{4}/;
     function testInfo(phoneInput) {
       var OK = re.exec(phoneInput.value);
       if (!OK)
         window.alert(RegExp.input + " isn't a phone number with area code!");
       else window.alert("Thanks, your phone number is " + OK[0]);
     }
    </script>
  </head>
  <body>
    <p>
      Enter your phone number (with area code) and then click "Check". <br />The
      expected format is like ###-###-####.
    </p>
    <form action="#">
      <input id="phone" /><button
       onclick="testInfo(document.getElementById('phone'));">
        Check
      </button>
    </form>
  </body>
</html>
```

This page was last modified on 7 авг. 2023 г. by MDN contributors.