8.1 Regular expressions

Introduction to regular expressions

Programs often need to determine if a string conforms to a pattern. Ex: A user is asked for their phone number, and the program must recognize if the input is formatted like a phone number. Or perhaps a program needs to search through a large collection of DNA sequences and replace defective gene sequences with correct sequences. Developers use regular expressions to solve these types of problems.

A ***regular expression*** (often shortened to ***regex***) is a string pattern that is matched against a string. Regular expressions may be defined with a ***RegExp*** object or between two forward slashes. Ex: `let re = new RegExp("abc");` or more simply: `let re = /abc/;` The pattern "abc" matches any string that contains "abc". Ex: "abcde" matches but "abd" does not. The RegExp method ***test(str)*** returns true if the string `str` matches the regex, and false otherwise.

PARTICIPATION ACTIVITY 8.1.1: Searching an array for the pattern 'ab'.

1 2 3 4 5 6 2x speed

```
let words = ["ban", "babble", "make", "flab"];

let re = /ab/;

words.forEach(function(word) {
   if (re.test(word)) {
      console.log(word + " matches!");
   }
});
```

babble matches!
flab matches!

re.test("flab") returns true because "flab" contains "ab".

Captions

1. Define a regular expression that matches words containing "ab".
2. Loop through each word in the words array.
3. re.test("ban") returns false because "ban" does not contain "ab".
4. re.test("babble") returns true because "babble" contains "ab".
5. re.test("make") returns false because "make" does not contain "ab".
6. re.test("flab") returns true because "flab" contains "ab".

Feedback?

PARTICIPATION ACTIVITY

8.1.2: Simple regular expressions.

1) The regex `/run/` matches the string "pruning".

- (•) True
- ( ) False

Correct

The characters "run" appear in consecutive order in "pruning".

2) The regex `/run/` matches the string "pr uning".

- ( ) True
- (•) False

Correct

The space between "r" and "un" causes the regex not to match.

3) What pattern would match the string "Regular Expression"?

- (•) Exp
- ( ) exp

Correct

Regular expressions are case sensitive, so "Exp" matches the capital "E" in "Expression".

4) What value is `x`?

```
let re = /regex/;
let x =
re.test("regular
expression");
```

- ( ) true
- (•) false

Correct

The characters "regex" do not appear in consecutive order in "regular expression".

5) What value is `x`?

```
let re = /c\/1/;
let x =
re.test("abc/123");
```

- (•) true
- ( ) false

Correct

Because the forward slash character indicates the beginning and the ending of a regex, the backslash character must precede the forward slash (like this: `\/`) to match a forward slash.

Feedback?

Special characters

Regular expressions use characters with special meaning to create more sophisticated patterns. The + character matches the preceding character at least once. Ex: `/ab+c/` matches one "a" followed by at least one "b" and one "c", so "abc" and "abbbbc" both match. However, "ac" does not match because the required "b" is missing.

Parentheses are used in a regex to match consecutive characters with *, +, and ?. Ex: `/(ab)+/` matches one or more "ab", so "abab" and "abbb" both match. However, "acb" does not match because the "c" is between "a" and "b".

Table 8.1.1: Selected special characters in regex patterns.

Character	Description	Example
`*`	Match the preceding character 0 or more times.	`/ab*c/` matches "abc", "abbbbc", and "ac".
`+`	Match the preceding character 1 or more times.	`/ab+c/` matches "abc" and "abbbbc" but not "ac".
`?`	Match the preceding character 0 or 1 time.	`/ab?c/` matches "abc" and "ac", but not "abbc".
`^`	Match at the beginning.	`/^ab/` matches "abc" but not "cab".
`$`	Match at the end.	`/ab$/` matches "cab" but not "abc".
|	Match string on the left OR string on the right.	`/ab|cd/` matches "abc" and "bcd".

Feedback?

PARTICIPATION ACTIVITY 8.1.3: Regex with special characters.

1) What string does NOT match the regex `/grea*t/`?

- `gret`
- `greaaat`
- `grat`

Correct

"grat" is missing an "e".

2) What string does NOT match the regex `/w+hy/`?

- `why`
- `hy` ✓
- `wwwwhy`

Correct

`w+` means at least one "w" must exist.

3) What string does NOT match the regex `/cat|bat|mat/`?

- `bbatt`
- `at` ✓
- `mat`

Correct

"at" does not match because "at" is not preceded by c, b, or m.

4) What regex matches the string `"boom"`?

- `/ba?oom/` ✓
- `/b\?oom/`
- `/bo?m/`

Correct

`a?` means the "a" may be missing.

5) What regex matches the string `"sleep like a baby"`?

- `/^like/`
- `/like$/`
- `/^sleep/` ✓

Correct

"sleep" is at the beginning of the string.

6) What regex matches the string `"that's easy"`?

- `/b?eas$/`
- `/b?sy$/` ✓
- `/e+sy$/`

Correct

`b?` matches a missing "b", and `sy$` matches "sy" at the end of the string.

7) What regex matches the string `"breaeak"`?

- `/bre+a+k/`
- `/br(ea)+k/` ✓
- `/er|bk/`

Correct

`(ea)+` matches the string "ea" one or more times.

8) What regex matches the string `"what?"`?

Correct

- ○ `/what?$/`
- ◉ `/what\?$/`
- ○ `/what$/`

A backlash before `?` matches a literal "?".

Feedback?

Character ranges

Square brackets are used in regular expressions to match any character in a range of characters. Ex: `/[aeiou]/` matches any vowel, and `/[0-9]/` matches any digit. The not operator (^) negates a range. Ex: `[^str]` matches any character except s, t, or r.

PARTICIPATION ACTIVITY 8.1.4: Regex with brackets.

Fill in the blank so the regex matches the description.

1) Match only the digits 0 through 5.
`/[____]/`

0-5

Check Show answer

Correct

0-5

Matches "4" but not "6".

2) Match only the letters a through f.
`/[____]/`

a-f

Check Show answer

Correct

a-f

Matches "b" but not "g".

3) Match anything except a vowel (a, e, i, o, u). List characters in the regex alphabetically.
`/[____]/`

^aeiou

Check Show answer

Correct

^aeiou

Matches "z" but not "e".

Feedback?

Metacharacters

A ***metacharacter*** is a character or character sequence that matches a class of characters in a regular expression. Ex: The period character matches any single character except the newline character. So `/ab.c/` matches "abZc" and "ab2c", but not "abc" since the period must match a single character. Other metacharacters begin with a backslash.

Table 8.1.2: Selected metacharacters in regex patterns.

Metacharacter	Description	Example
`.`	Match any single character except newline.	`/a.b/` matches "aZb" and "a b".
`\w`	Match any word character (alphanumeric and underscore).	`/a\wb/` matches "aAb" and "a5b" but not "a b".
`\W`	Match any non-word character.	`/a\Wb/` matches "a-b" and "a b" but not "aZb".
`\d`	Match any digit.	`/a\db/` matches "a2b" and "a9b", but not "aZb".
`\D`	Match any non-digit.	`/a\Db/` matches "aZb" and "a b", but not "a2b".
`\s`	Match any whitespace character (space, tab, form feed, line feed).	`/a\sb/` matches "a b" but not "a4b".
`\S`	Match any non-whitespace character.	`/a\Sb/` matches "a!b" but not "a b".

Feedback?

PARTICIPATION ACTIVITY 8.1.5: Metacharacters in regex.

Match the regex to a string the regex matches.

If unable to drag and drop, refresh the page.

`/1\w+/`	123break `1` matches "1" and `\w+` matches "23break".	Correct
`/9\d+/`	923 break `\d+` matches 23.	Correct
`/1\s\d/`	break1 9 `\s\d` matches " 9".	Correct
`/\W\W\D/`	()break `\W\W` matches "()", and `\D` matches "b".	Correct
`/\d\S\?/`	5!?5break `\d` matches "5", `\S` matches "!", and `\?` matches "?".	Correct
`/\d\./`	0.break `\d` matches "0" and `\.` matches ".".	Correct

Reset

Feedback?

Mode modifiers

A ***mode modifier*** (sometimes called a ***flag***) changes how a regex matches and is placed after the second slash in a regex. Ex: `/abc*/i` specifies the mode modifier `i`, which performs case-insensitive matching.

Table 8.1.3: Selected mode modifiers.

Mode modifier	Description	Example
`i`	Case insensitivity - Pattern matches upper or lowercase.	`/aBc/i` matches "abc" and "AbC".
`m`	Multiline - Pattern with `^` and `$` match beginning and end of any line in a multiline string.	`/^ab/m` matches the second line of "cab\nabc", and `/ab$/m` matches the first line.

`g`	Global search - Pattern is matched repeatedly instead of just once.	`/ab/g` matches "ab" twice in "cababc".

Feedback?

PARTICIPATION ACTIVITY 8.1.6: Regex with mode modifiers.

1) What string does NOT match the regex `/great/i`?

- `Great`
- `greaT`
- `TREAT`

Correct

The `i` mode uses case insensitive matching. "TREAT" must be "gREAT" or "GREAT" to match.

2) What string does NOT match the regex `/ly$/m`?

- `quick most first`
- `quick mostly first`
- `quick most firstly`

Correct

The `m` mode uses multiline matching, and none of the lines end with "ly".

3) How many times does the regex `/moo/g` find a match in "Moo the cow mooed at the moon."

- 1
- 2
- 3

Correct

The `g` mode uses global matching, so both appearances of "moo" in "mooed" and "moon" are matched.

4) How many times does the regex `/moo/gi` find a match in "Moo the cow mooed at the moon."

- 1
- 2
- 3 (selected)

Correct

The `g` (global matching) and `i` (case insensitive) modes are used together. The regex finds the pattern `moo` in "Moo", "mooed", and "moon".

Feedback?

PARTICIPATION ACTIVITY 8.1.7: Using regular expressions to identify secret messages.

A criminal organization is using Reddit to communicate. To keep from being detected, the criminals are posting comments that look innocuous but use a secret pattern.

- The pattern contains one or more digits followed by any number of characters, followed by the word "star". Ex: "3stars" and "99 bright stars!" should both match.
- The letters in the word "star" may be separated by a single space. Ex: "1 blast ark" and "1 s t a r" should match.
- The comments can include upper or lowercase characters. Ex: "2 STar" should match.

Loop through the Reddit posts in the `posts` array and output to the console the lines that match the criminal's pattern. Use a single regex to identify the suspected posts. Hint: The 2nd, 3rd, and 5th lines should match the regex.

```
let posts = [
   "The starting time was 6pm.",
   "If the 2nd string QB gets hurt, they have no starting QB.",
   "My email is sis1@yahoo.com.  Last are first.",
   "Stare into the abyss 1 time.",
   "90210 for Beverly Hills. Thick as TAR."
];

// Modify to output only lines that match regex
posts.forEach(function(line) {
   console.log(line);
});

```

Run JavaScript Reset code

Your console output

```
The starting time was 6pm.
If the 2nd string QB gets hurt, they have no starting QB.
My email is sis1@yahoo.com.  Last are first.
Stare into the abyss 1 time.
90210 for Beverly Hills. Thick as TAR.
```

▶ View solution

Feedback?

CHALLENGE ACTIVITY 8.1.1: Regular expressions.

530096.4000608.qx3zqy7

Jump to level 1

1 2 3 4

Assign re with a regular expression that verifies a phone number with an optional leading 1. Ex: 1-800-555-2468 or 800-555-2468.

```
1 let re = /^1?-?\d{3}-\d{3}-\d{4}$//* Your solution goes here */;
```

1 2 3 4

Check Try again

Done. Click any level to practice more. Completion is preserved.

✔

✔ Testing match with string "1-800-555-2468"

Yours `true`

✔ Testing match with string "800-555-2468"

Yours `true`

✔ Testing match with string "800-555-246"

Yours `false`

✔ Testing match with string "8888888888"

Yours `false`

✔ Testing match with string "101-303-777-6666"

Yours `false`

✔ Testing match with string "1-303-777-6666001"

Yours `false`

Feedback?

Determining what matches

The RegExp method ***exec(str)*** determines what part of the string `str` matches a regex. The `exec()` method returns a result array, or returns `null` if the pattern does not match.

Figure 8.1.1: Using exec() to discover what characters matched the regex.

```
let re = /t.+r/;
let result = re.exec("Raise the bar
high.");

if (result === null) {
   console.log("No match.");
}
else {
   // Index 0 is the full string that
matched
   console.log(result[0]);   // the bar
}
```

Feedback?

Parentheses in a regex are used to "remember" different parts of a matched string. Ex: `/a(b+)c/` remembers anything matching `(b+)`, so "bbb" is remembered when applying the regex to "abbbc". The remembered parts are accessible from the result array returned by `exec()`. The first array element is the complete matched string, and the following elements are the remembered parts. If the regex contains no parentheses, the returned array contains only the complete string that matches.

Figure 8.1.2: Remembering matches in a regex.

```
let re = /(B.+)'s (.+day)/;
let result = re.exec("When is Becky's
birthday?");

// Index 0 is the full string that matched
console.log(result[0]);   // Becky's birthday

// Index 1 is the first remembered part
console.log(result[1]);   // Becky

// Index 2 is the second remembered part
console.log(result[2]);   // birthday
```

Feedback?

PARTICIPATION ACTIVITY 8.1.8: Extracting regex matches.

Twitter wants to know which hashtags are currently trending and what websites are tweeted most often. A selection of tweets are given in the `tweets` array. Create two regular expressions that will:

1. Extract all the hashtags used in the tweets. A hashtag begins with a pound sign and contains all following word characters. Ex: #myHashTag. Output each hashtag to the console.
2. Extract all the domain names from the URLs in the tweets. A URL begins with a protocol and double slash: "http://" or "https://". The domain name is the string of characters immediately after the double slash and before the next forward slash (/). Ex: The domain name for `https://en.wikipedia.org/wiki/URL` is `en.wikipedia.org`. Output each domain name to the console.

Multiple hashtags and URLs may exist in a single tweet, so use the "g" mode modifier on both regexes and loop until the pattern is no longer found. To extract the domain name, use `.+?` to match the characters after the double slash and before the first slash. The `+?` operator is "lazy" and matches as few characters as possible, whereas + matches as many characters as possible.

```
const tweets = [
   "Thank you to the Academy and the incredible cast & crew of #
   "@HardingCompSci department needs student volunteers for #Hou
   "Checkout the most comfortable earbud on #Kickstarter and boo
   "Bootstrap 4 Cheat Sheet https://t.co/MFyKHvd50H",
   "Curious to see how #StephenCurry handles injury. http://mash
];

// Extract hashtags and domain names from URLs
tweets.forEach(function(tweet) {
   console.log(tweet);
});
```

Run JavaScript Reset code

Your console output

```
Thank you to the Academy and the incredible cast & crew of #TheRevenant. ;
@HardingCompSci department needs student volunteers for #HourOfCode https
Checkout the most comfortable earbud on #Kickstarter and boost your #produ
Bootstrap 4 Cheat Sheet https://t.co/MFyKHvd50H
Curious to see how #StephenCurry handles injury. http://mashable.com/2016/
```

▶ View solution

Feedback?

PARTICIPATION ACTIVITY 8.1.9: The exec() method.

What is output to the console?

1)

```
re = /\w+y/;
result = re.exec("zing
zany zone");
console.log(result[0]);
```

- ○ zing
- ○ zing zany
- ◉ zany

Correct

The full match contains all consecutive word characters that end in "y".

2)

```
re = /\w+y/;
result =
re.exec("zing zane
zone");
console.log(result);
```

- ○ undefined
- ◉ null
- ○ false

Correct

The string has no "y", so `exec()` returns null.

3)

```
re = /\w+y/;
result = re.exec("zing
zany zone");
console.log(result[1]);
```

- (●) undefined
- ( ) null
- ( ) false

Correct

No parenthesis exist in the regex, so no string values are placed in the result array except `result[0]`, which contains the full string match.

4)

```
re = /z(\w+)/;
result = re.exec("zing
zany zone");
console.log(result[1]);
```

- ( ) zing
- (●) ing
- ( ) ing zany zone

Correct

`result[1]` contains the first "remembered" match in parenthesis.

5)

```
re = /z\w+/g;
str = "zing zany zone";
result = re.exec(str);
while (result !== null)
{

console.log(result[0]);
   result =
re.exec(str);
}
```

- ( ) zing
- (●) zing
zany
zone
- ( ) Infinite loop

Correct

The "g" mode modifier at the end of the regex causes `exec()` to look for multiple matches in the same string.

6)

```
re = /\d+?/;
str = "0123";
result = re.exec(str);
console.log(result[0]);
```

- (●) 0
- ( ) 3
- ( ) 0123

Correct

`+?` matches the fewest characters, which is 0.

Feedback?

CHALLENGE ACTIVITY 8.1.2: Determining what matches.

530096.4000608.qx3zqy7

Jump to level 1

1

Write a statement that creates a RegExp object that remembers the title and first name, but does not remember the last name. Using the RegExp object's exec() method, assign the result variable with the remembered title and first name of the string userName.

2

3

```
1 const userName = "Dr. Greg House"; // Code will also be tested wi
2
3 /* Your solution goes here */
4 const re = /(Dr\.|Mr\.)\s(\w+)/;
5
6 const result = re.exec(userName);
7
8
9 console.log(result[1] + " " + result[2]);
```

1 2 **3**

Check Next **Done**. Click any level to practice more. Completion is preserved.

✔

✔ Testing displayed output with userName = "Dr. Greg House"

Yours `Dr. Greg`

✔ Testing displayed output with userName = "Mr. Howard Wolowitz"

Yours `Mr. Howard`

Feedback?

String methods that use regex

Several String methods work with regular expressions:

- ***match()*** *returns an array of the matches made when matching the string against a regex.*
- ***replace()*** *returns a new string that replaces matching strings with a replacement string.*
- ***search()*** *returns the index of the first match between the regex and the given string, or -1 if no match is found.*

- ***split()*** *returns an array of strings created by separating the string into substrings based on a regex.*

Exploring further:

- Regular Expressions (MDN)
- RegExr - For testing regular expressions

How was this section? Provide section feedback