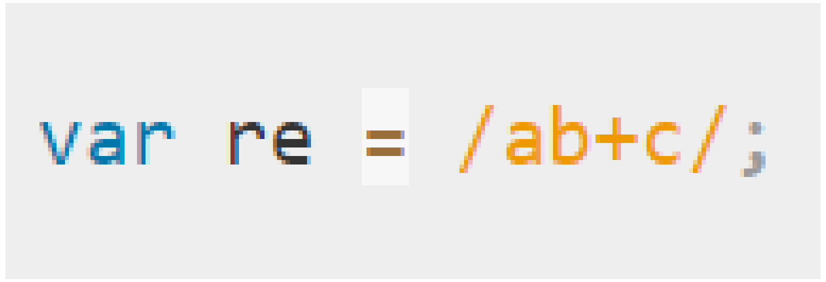
JAVASCRIPT & REGEX

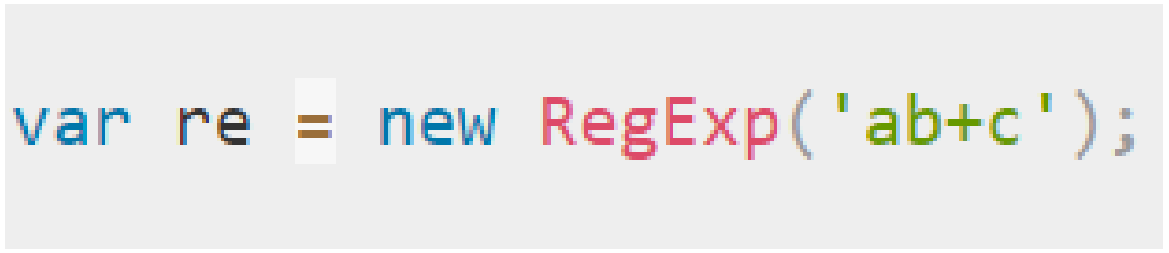
REGULAR EXPRESSIONS (REGEX)

- Used to test and match character combinations in strings
- In javascript, regexes are also objects
- Object is called – RegExp
- RegExp methods – `exec & test`
- String methods – `match, matchAll, replace, search, split`

TWO WAYS TO CREATE



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



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

WHAT IS A REGEX

- Pattern composed of:
  - Simple characters such as `/abc/`
  - Combination of simple and special characters,
such as `/ab*c/` or `/Week (\d+)\. \d*/`

SIMPLE PATTERNS

- Characters for which you want to find a direct match
- Matches only when the characters occur in the exact same sequence
- So `/abc/` matches

`“Do you know your abcs?”` and

`“Slabcraft is a forgotten way to design airplanes”`

But will not match

`“Grab crab”`

USING SPECIAL CHARACTERS

- When you need more than a simple match (e.g., space, number of matches, etc.)

- In pattern `/ab*c/`, the * matches 0 or more of the preceding characters
- Will match

‘`abbbbc`’ inside ‘`cbbabbbbcdebc`’

SPECIAL CHARACTERS - I

\	If it precedes a non-special character, indicates next character is special and not to be interpreted E.g. \b ➔ word boundary If it precedes a special character, indicates next character is not special and should be interpreted literally E.g., /a*b/ → looks for the literal string 'a*b'
^	Matches beginning of input. If multiline is set to true, also matches immediately after linebreak E.g., ^A → "An empty field" but does not match "cAts are not funnAy"
$	Matches end of input. If multiline is set to true, also matches immediately before linebreak. E.g., /t$/ → matches "eat", but does not matching "eating"

SPECIAL CHARACTERS – II

*	Matches the preceding character 0 or more times. Equivalent to {0,} E.g., `/bo*t/` → matches "boot" "bot", "bt" but does not match "bit"
+	Matches the preceding character 1 or more times. Equivalent to {1,} E.g., `/a+/` → matches "candy" and "caaaaaandy" but does not match "cindy"
?	Matches the preceding character 0 or 1 times. Equivalent to {0,1} `/e?le?/` → matches both "angel" and "angle" If used immediately after a quantifier (*,+,?,{}), makes the match non-greedy. E.g., `/\d+/` matches "123" but /\d+?/ matches only "1"
.	Matches any character except for newline `/.n/` → matches "an" "in" and "on". Does not match "no"
|	Matches x | y. Also called Alternation E.g., `/green|red/` matches both "green apple" as well as "red apple"

SPECIAL CHARACTERS – III

(x)	Matches 'x' and remembers it. These are called capturing parantheses `"foo bar".replace(/(…) (…)/, '$2 $1')` → results in "`bar foo`"
(?:x)	Matches 'x' but does not remember it. These are called non-capturing parantheses `/(?:foo){1,2}/` → applies to the whole word – `foo`.
x(?=y)	Matches 'x' only if it is followed by 'y'. This is also called lookahead `/Jack(?=Black)/` → only matches "`Jack Black`" but not "`Jack Sprat`"
x(?!y)	Matches 'x' only if it *not* followed by 'y'. This is also called negated lookahead `/\d+(?!\.)/` → only matches a number if it is not followed by a decimal point
(?<=y)x	Matches 'x' only if preceded by 'y'. This is also called a lookbehind `/(?<=Jack)Sprat/` only matches "Sprat" if it is preceded by "Jack". So 'Jack Sprat' matches but not 'Tom Sprat'
(?<!y)x	Matches 'x' only if *not* preceded by 'y'. This is also called a negated lookbehind `/(?<!-)\d+/` matches a number only if it is not preceded by a minus sign

SPECIAL CHARACTERS – IV

{n}	Match exactly ‘n’ occurrences of the preceding expression. N must be positive integer /a{2}/ will not match “candy” but will match “caandy” as well as “caaaaandy”
{n,}	Match at least ‘n’ occurrences of the preceding expression. N must be a positive integer /a{2, }/ will match “aa” and “aaaaa” but will not match “a”
{n,m}	Match at least ‘n’ and at most ‘m’ occurrences of the preceding expression. n <= m /a{2,3}/ will not match “candy” but will match “candy” and “caaaaandy”
[xyz]	Matches the character set [a-d] will match [abcd]. [0-9] will match [0123456789]
[^xyz]	Negated matching of characters [^a-d] will match anything that is not [abcd]

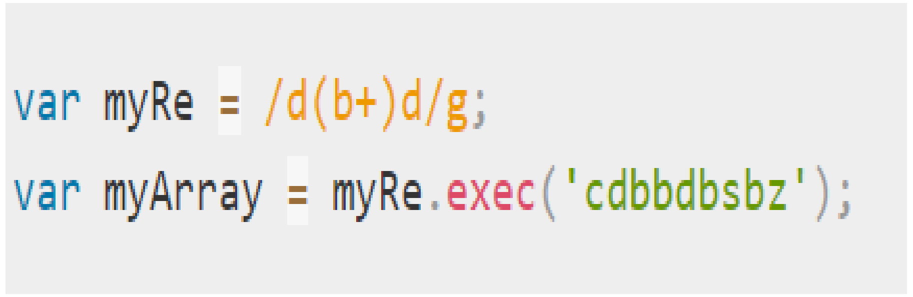
SPECIAL CHARACTERS – V

\b	Matches a word boundary /\bm/ will match "moon" because 'm' is at the word boundary
\B	Matches a non-word boundary /\B../ will match 'oo' in "noon"
\d	Matches a digit. Equivalent to [0-9]
\D	Matches a non-digit character. Equivalent to [^0-9]
\n	Matches a linefeed
\s	Matches a space
\S	Matches a non-space character
\w	Matches any alphanumeric character. Equivalent to [A-Za-z0-9_]
\W	Matches any non-word character. Equivalent to [^A-Za-z0-9_]

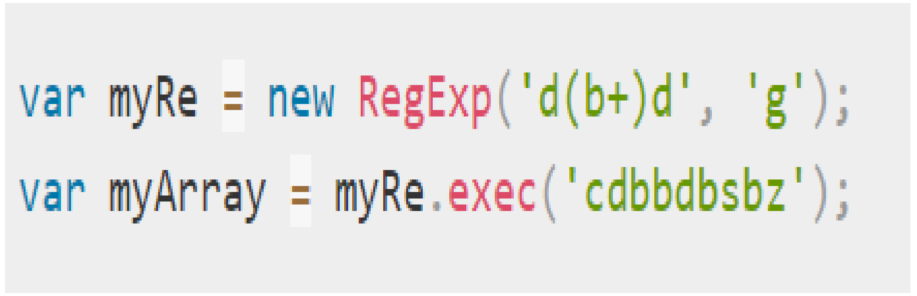
WORKING WITH REGEX – I

- RegExp object

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



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



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

CHECK IF A PATTERN EXISTS

```
let re = /[a-z]+/;

if (re.test("foo")) {

    console.log("Match exists.");

}
```

MATCHING WITH .EXEC

`exec` returns an array of captures or `null` if there was no match

```
let re = /([0-9]+)[a-z]+/;
```

```
let match = re.exec("foo123bar");
```

`match.index` is 3, the (zero-based) location of the match.

`match[0]` is the full match string.

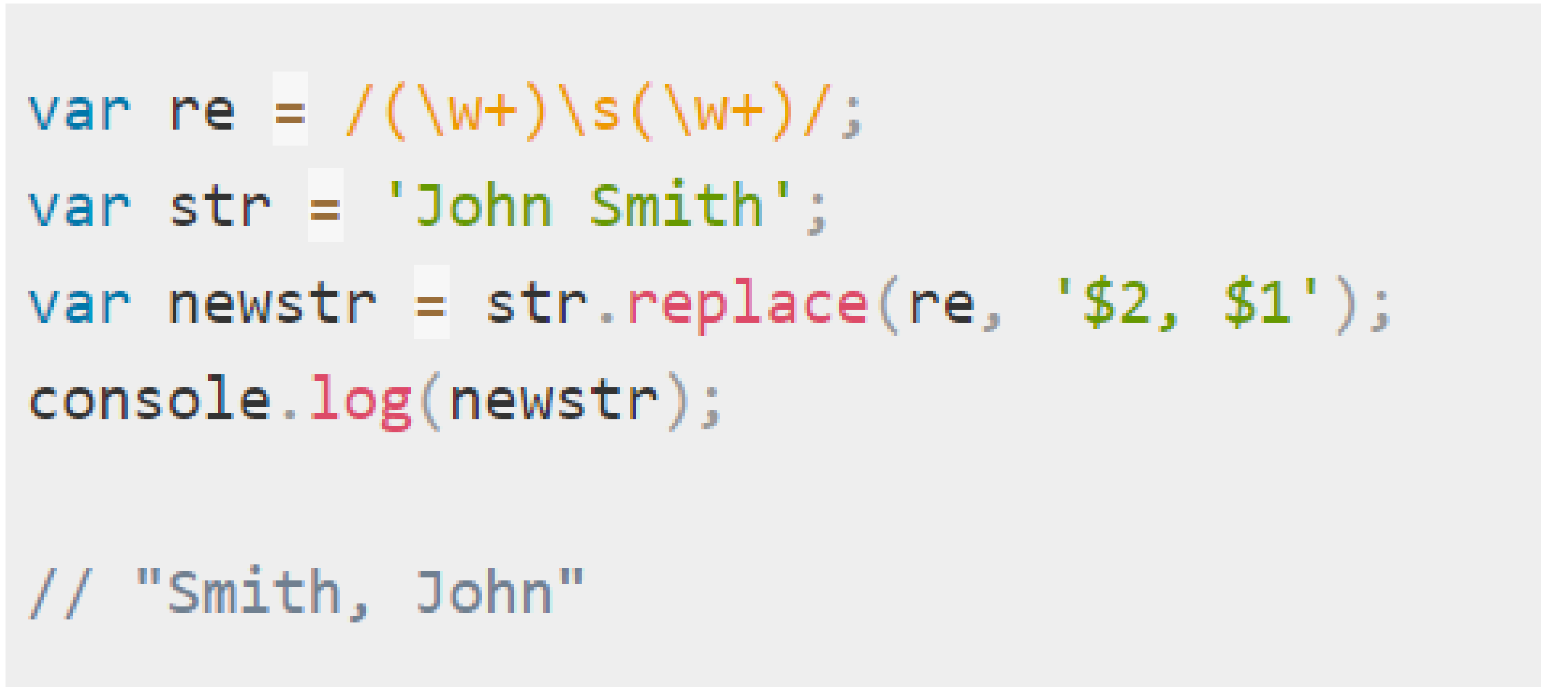
`match[1]` is the text corresponding to the first captured group. `match[n]` would be the value of the nth captured group

WORKING WITH REGEX – II

- String Object

match	A String method that returns an array containing all of the matches, including capturing groups, or null if no match is found.
matchAll	A String method that returns an iterator containing all of the matches, including capturing groups.
search	A String method that tests for a match in a string. It returns the index of the match, or -1 if the search fails.
replace	A String method that executes a search for a match in a string, and replaces the matched substring with a replacement substring.
split	A String method that uses a regular expression or a fixed string to break a string into an array of substrings

EXAMPLE WITH STRING

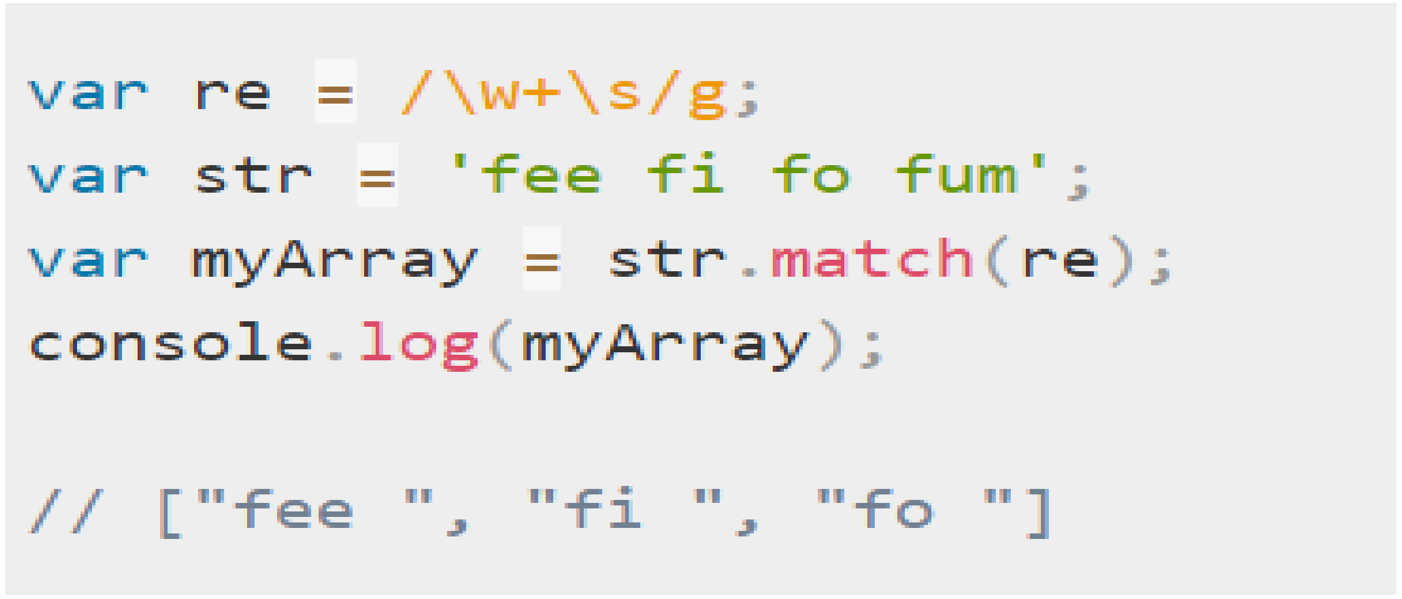


```
var re = /(\w+)\s(\w+)/;
var str = 'John Smith';
var newstr = str.replace(re, '$2, $1');
console.log(newstr);

// "Smith, John"
```

SEARCHING WITH FLAGS

g	Global search
i	Case-insensitive search
m	Multi-line search



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

// ["fee ", "fi ", "fo "]
```

LOOP THROUGH MATCHES

- Using exec()

```
let re = /a/g;

let result;

while ((result = re.exec('barbatbaz')) !== null) {

    console.log("found '" + result[0] + "', next exec starts at index '" + re.lastIndex + "'");

}
```

Expected Output

```
found 'a', next exec starts at index '2'

found 'a', next exec starts at index '5'

found 'a', next exec starts at index '8'
```

COMBINE STRING WITH REGEXP

- `"string".match(...)`
- `"string".replace(...)`
- `"string".split(...)`
- `"string".search(...)`

```
console.log("string".match(/[i-n]+/));

console.log("string".match(/(r)[i-n]+/));
```

Expected Output

```
Array ["in"]

Array ["rin", "r"]
```

REPLACE, SPLIT

```
console.log("string".replace(/[i-n]+/, "foo"));
```

Expected Output

```
Strfoog
```

```
console.log("stringstring".split(/[i-n]+/));
```

Expected Output

```
Array ["str", "gstr", "g"]
```

SEARCH

- Returns an index, if found. Else, -1

```
console.log("string".search(/[i-n]+/));

console.log("string".search(/[o-q]+/));
```

Expected Output

```
 3

-1
```

USING CONSOLE

- The console has multiple functions that can be useful for keeping time
  - `console.time(“some string”);`
  - `console.timeEnd(“some string”);`

- Console can also be used for grouping of messages
  - `console.group()`
  - `console.groupCollapsed();`
  - `console.groupEnd();`

OTHER CONSOLE METHODS

- `console.info` – small informative icon (ⓘ) appears on the left side of the printed string(s) or object(s).
- `console.warn` – small warning icon (!) appears on the left side. In some browsers, the background of the log
- is yellow.
- `console.error` – small times icon (⊗) appears on the left side. In some browsers, the background of the log is red.
- `Console.table` – display objects or arrays in a tabular format

OTHER CONSOLE METHODS

- `console.clear()` - This removes all previously printed messages in the console
- `console.dir(object)` - displays an interactive list of the properties of the specified JavaScript object. The output is presented as a hierarchical listing with disclosure triangles that let you see the contents of child objects.
- `console.assert()` – useful for debugging.

Beware! Code does NOT stop executing!

CLASS EXERCISE

- Consider the following string

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

Use regular expressions and the String methods `split()` and `replace()` to produce output like this

```
// ---------- Sorted
// Abel, Bill
// Barney, Fred
// Carrot, Orange
// Hand, Chris
// Rigby, Helen
// ---------- End
```

GO THROUGH THIS TUTORIAL

- https://javascript.info/regexp-introduction