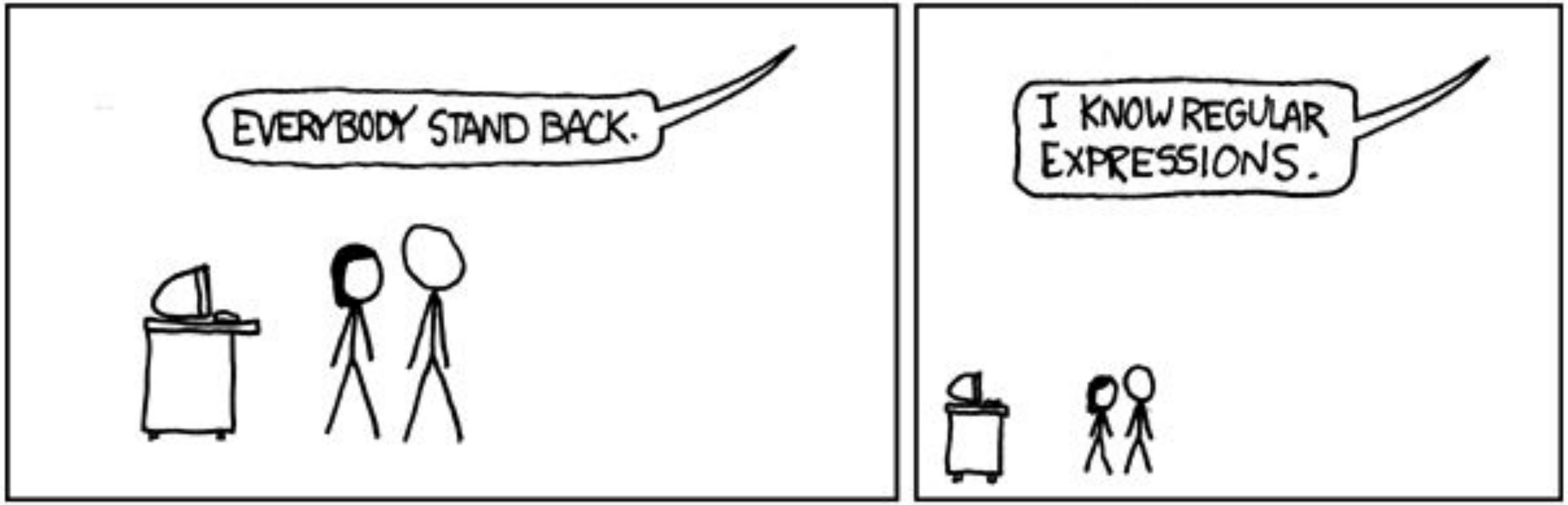
^Regular\sExpressions$



Lakusta Valeria,
Computational Linguist at Grammarly

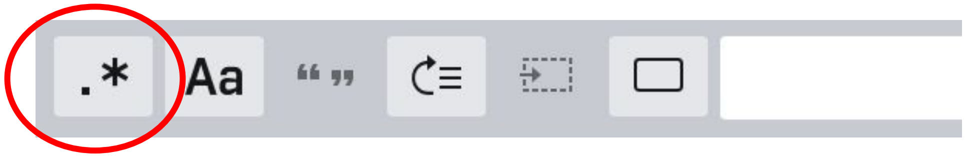
Powerful tool for matching patterns

HOW TO REGEX
STEP 1: OPEN YOUR FAVORITE EDITOR
@GARABATOKID
STEP 2: LET YOUR CAT PLAY ON YOUR KEYBOARD
/^([A-Z0-9_\.-

Sublime Text

Find: Command + f
Find & replace: Option + command + f

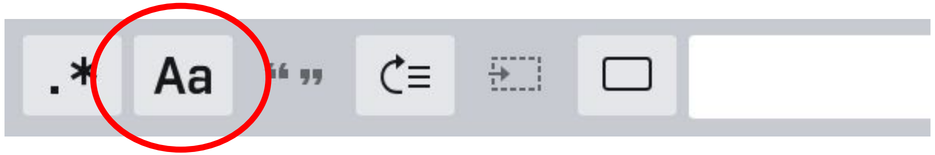
Find: Alt + Command + f
Find & replace: Alt + Command + f



Sublime Text

Find: Command + f
Find & replace: Option + command + f

Find: Alt + Command + f
Find & replace: Alt + Command + f



Basic concepts

Basic concepts

1. Characters

2. Operations

Basic concepts

1. Characters
● ordinary

2. Operations

“A” “a” “Hello” 12 0

Basic concepts

1. Characters
- ordinary
- metacharacters

2. Operations

“A” “a” “Hello” 12 0

^ $. ( ) [] \

Basic concepts

1. Characters
- ordinary
- metacharacters

2. Operations
- quantification

“A” “a” “Hello” 12 0

^ $. ( ) [] \

+ ? * {...}

Basic concepts

1. Characters
- ordinary
- metacharacters

2. Operations
- quantification
- grouping

“A” “a” “Hello” 12 0

^ $. ( ) [] \

\+ ? * {...}

Quantifiers

Quantifiers should be placed after the part of the regular expression that you want to quantify.

Quantifiers

Quantifiers should be placed after the part of the regular expression that you want to quantify.

? - zero or one
***** - zero or more
+ - one or more

Quantifiers

Quantifiers should be placed after the part of the regular expression that you want to quantify.

? - zero or one

***** - zero or more

+ - one or more

a+ matches one or more "a"s, like "a", "aaa" or "aaaaaaaaa"

cats? matches two strings: "cat" and "cats"

Quantifiers

Quantifiers should be placed after the part of the regular expression that you want to quantify.

{n} - exactly n times
{n1,n2} - n1 to n2 times
{n,} - n or more times

o{2} - matches “oo” in “balloon”

o{2,4} - matches “ooo” in “sooo beautiful!”

a{5,} - matches the string “aaaaaaaaa”

Quantifiers

Quantifiers should be placed after the part of the regular expression that you want to quantify.

If you want to apply a quantifier to a set of characters, use parenthesis.

cats? matches two strings: "cat" and "cats"

will (not)? matches “will not” and “will ”

Quantifiers

Quantifiers should be placed after the part of the regular expression that you want to quantify.

If you want to apply a quantifier to a set of characters, use parenthesis.

For one character

cats? matches two strings: "cat" and "cats"

will (not)? matches “will not” and “will ”

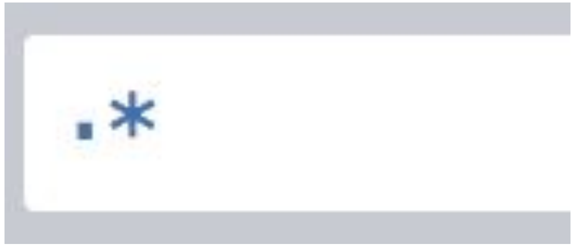
For a set of characters

Metacharacters

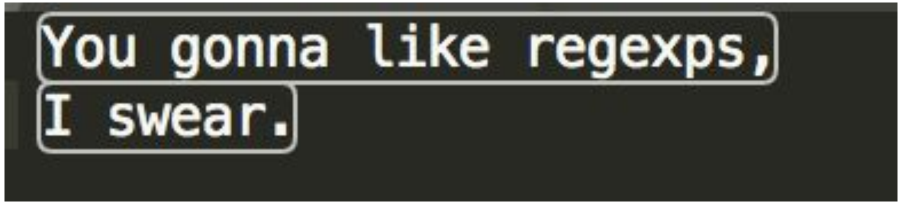
Metacharacters

. (Period)

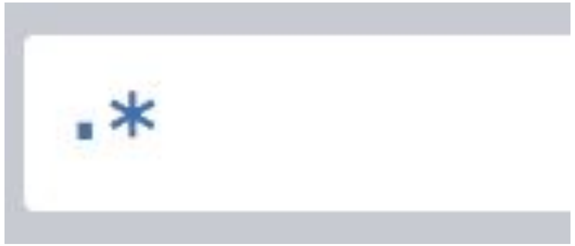
Matches everything except new line character \n !



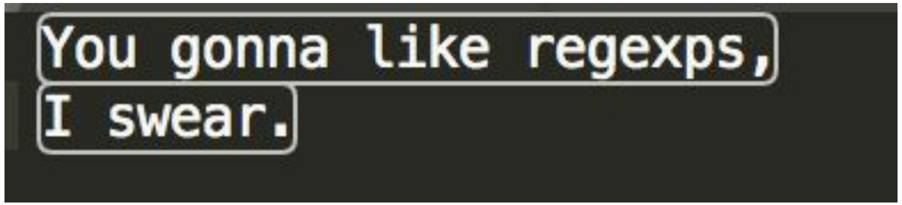
```
.*
```



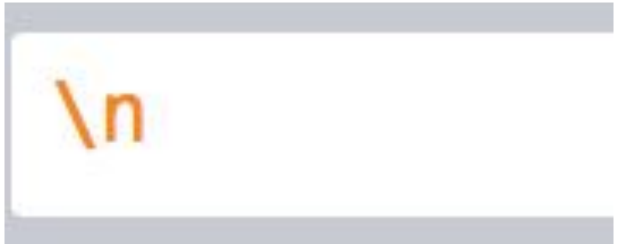
```
You gonna like regexps,
I swear.
```



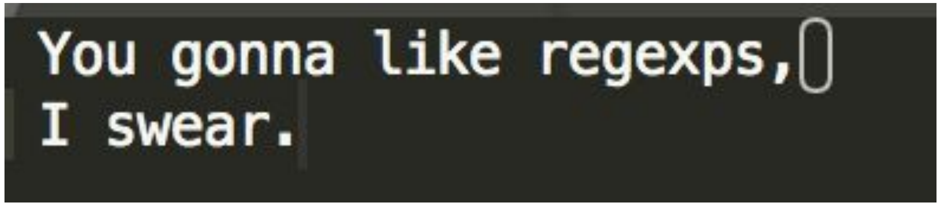
```
.*
```



```
You gonna like regexps,
I swear.
```



```
\n
```



```
You gonna like regexps,
I swear.
```

Practice time

Write a regex that matches both “king” and “kong” words

Practice time

Write a regex that matches both “king” and “kong” words

The answer is
“k.ng”

Metacharacters

Boundaries:

Quite often, you will need to match a string with certain boundaries.

Metacharacters

Boundaries:

Quite often, you will need to match a string with certain boundaries.

\b - word boundary, which stands for a position where
only one side is a letter, a digit or an underscore.
\B - not a word boundary

Metacharacters

Boundaries:

Quite often, you will need to match a string with certain boundaries.

\b - word boundary, which stands for a position where only one side is a letter, a digit or an underscore.
\B - not a word boundary

\bme\b matches the string "me" in " me"

\bme\B matches the string "me" in " meow "

Metacharacters

Boundaries:

Quite often, you will need to match a string with certain boundaries.

\\< - word boundary on the left
\\> - word boundary on the right

\\<me\\> matches the string "me" in " me"

Metacharacters

Boundaries:

Quite often, you will need to match a string with certain boundaries.

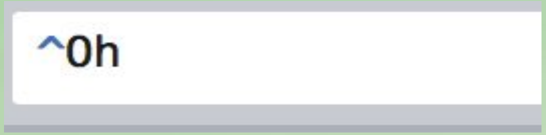
^ - the start of a string or a line
$ - the end of a string or a line

Metacharacters

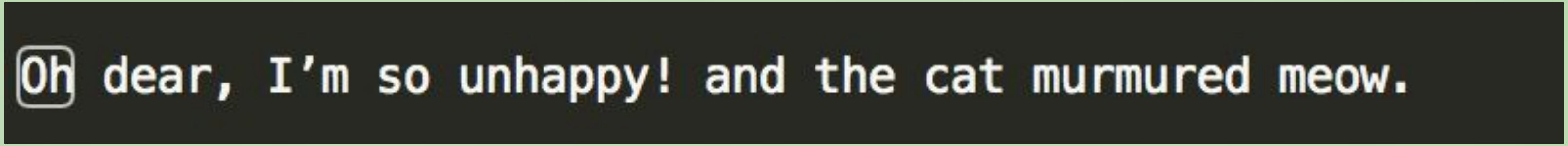
Boundaries:

Quite often, you will need to match a string with certain boundaries.

^ - the start of a string or a line
$ - the end of a string or a line



```
^Oh
```



```
Oh dear, I’m so unhappy! and the cat murmured meow.
```

Metacharacters

Boundaries:

Quite often, you will need to match a string with certain boundaries.

\A - the start of a string (aka the whole document)
\Z - the end of a string (aka the whole document)

Metacharacters

Boundaries:

Quite often, you will need to match a string with certain boundaries.

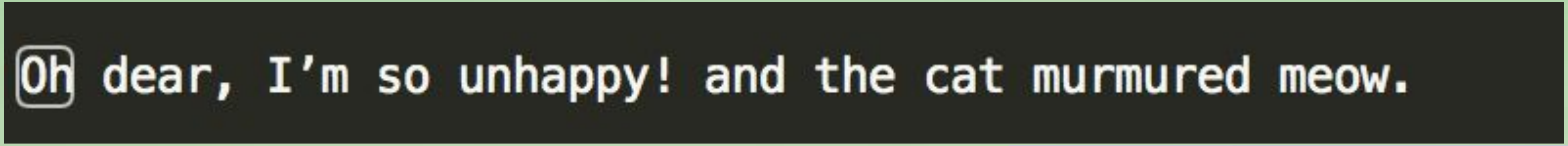
\A - the start of a string (aka the whole document)
\Z - the end of a string (aka the whole document)

Metacharacters

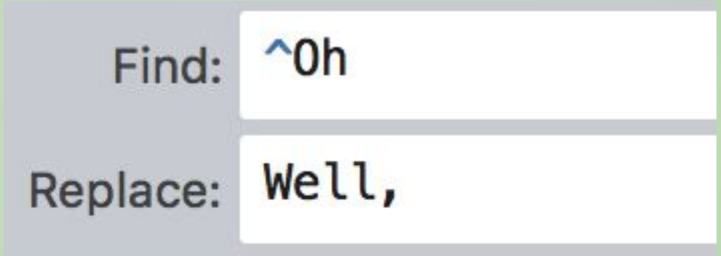
Boundaries:

Quite often, you will need to match a string with certain boundaries.

\A - the start of a string (aka the whole document)
\Z - the end of a string (aka the whole document)



```
Oh dear, I’m so unhappy! and the cat murmured meow.
```



Find: `^Oh`

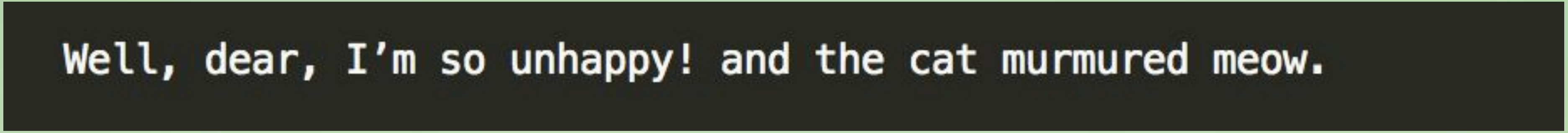
Replace: `Well,`

Metacharacters

Boundaries:

Quite often, you will need to match a string with certain boundaries.

\A - the start of a string (aka the whole document)
\Z - the end of a string (aka the whole document)



```
Well, dear, I'm so unhappy! and the cat murmured meow.
```

Boundaries don't match any characters. They are just boundaries.

Boundaries don't match any characters. They are just boundaries.

The caret ^ has a special meaning when it's used within square brackets.

Practice time

Write a regex that matches a word that consists of 3 letters and starts with “sa” .

Practice time

Write a regex that matches a word that consists of 3 letters and starts with “sa” .

The answers is
“\bsa.\b”

Metacharacters

Character Classes

Special symbols that match classes of characters.
These symbols are written with a backslash (\).

Metacharacters

Character Classes

Special symbols that match classes of characters.
These symbols are written with a backslash (\).

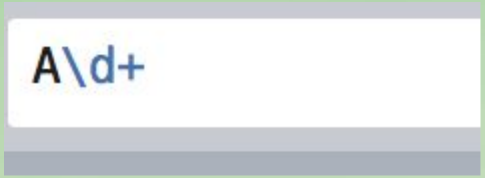
\d - one digit
\D - one non-digital character
\w - a letter, a digit or an underscore
\W - one character that is not a letter, a digit or an underscore

Metacharacters

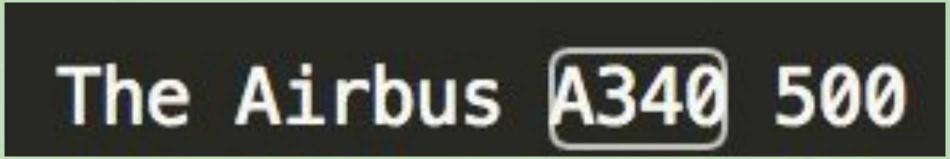
Character Classes

Special symbols that match classes of characters.
These symbols are written with a backslash (\).

\d - one digit
\D - one non-digital character
\w - a letter, a digit or an underscore
\W - one character that is not a letter, a digit or an underscore



```
A\d+
```



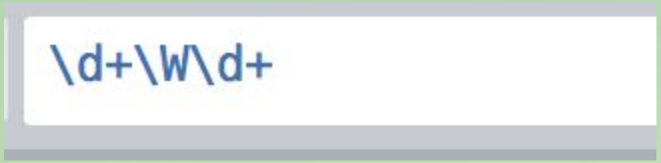
```
The Airbus A340 500
```

Metacharacters

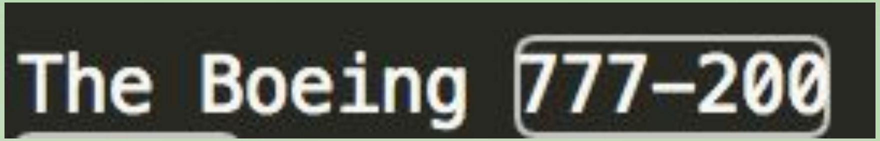
Character Classes

Special symbols that match classes of characters.
These symbols are written with a backslash (\).

\d - one digit
\D - one non-digital character
\w - a letter, a digit or an underscore
\W - one character that is not a letter, a digit or an underscore



```
\d+\W\d+
```



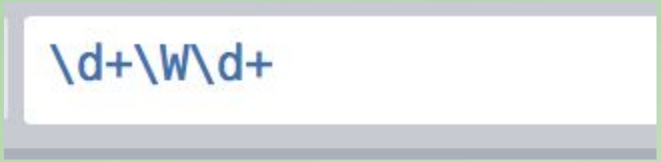
```
The Boeing 777-200
```

Metacharacters

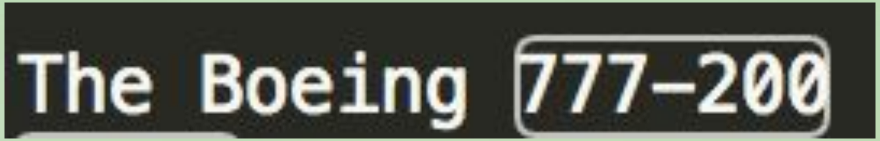
Character Classes

Special symbols that match classes of characters.
These symbols are written with a backslash (\).

\d - one digit
\D - one non-digital character
\w - a letter, a digit or an underscore
\W - one character that is not a letter, a digit or an underscore



```
\d+\W\d+
```



```
The Boeing 777-200
```

\W matches spaces or punctuation marks, for example "-"

Metacharacters

Character Classes

Special symbols that match classes of characters.
These symbols are written with a backslash (\).

\s - a whitespace character: a space, a tab, a newline, etc.
\S - one character that is not a whitespace character
\t - a tab
\n - a newline character

Metacharacters

Character Classes

Special symbols that match classes of characters.
These symbols are written with a backslash (\).

\s - a whitespace character: a space, a tab, a newline, etc.
\S - one character that is not a whitespace character
\t - a tab
\n - a newline character

\S+\s\S+ matches any two words separated with a space, like “about me”, “look at”, “hello, mom”, etc.

Metacharacters

Character Ranges

Regular expressions allow specifying ranges of characters with the help of square brackets.

Metacharacters

Character Ranges

Regular expressions allow specifying ranges of characters with the help of square brackets.

[...] - one of the characters in the brackets
[char1-char2] - one of the characters in the range from char1 to char2.

Metacharacters

Character Ranges

Regular expressions allow specifying ranges of characters with the help of square brackets.

[...] - one of the characters in the brackets
[char1-char2] - one of the characters in the range from char1 to char2.

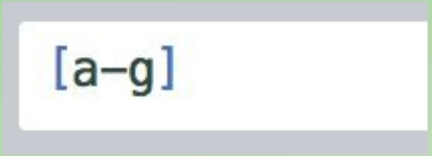
[aeuoi] matches one letter that represents a vowel sound, like "u"

Metacharacters

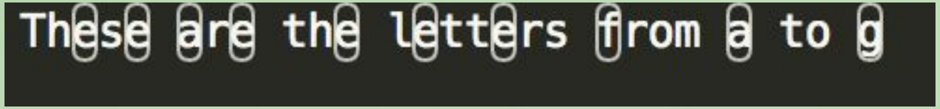
Character Ranges

Regular expressions allow specifying ranges of characters with the help of square brackets.

[…] - one of the characters in the brackets
[char1-char2] - one of the characters in the range from char1 to char2.



```
[a-g]
```



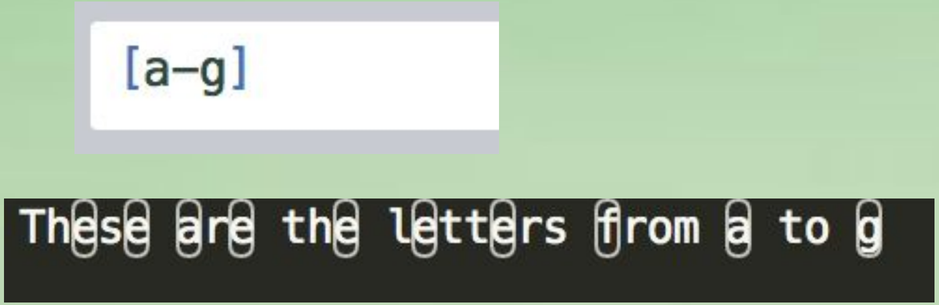
```
These are the letters from a to g
```

Metacharacters

Character Ranges

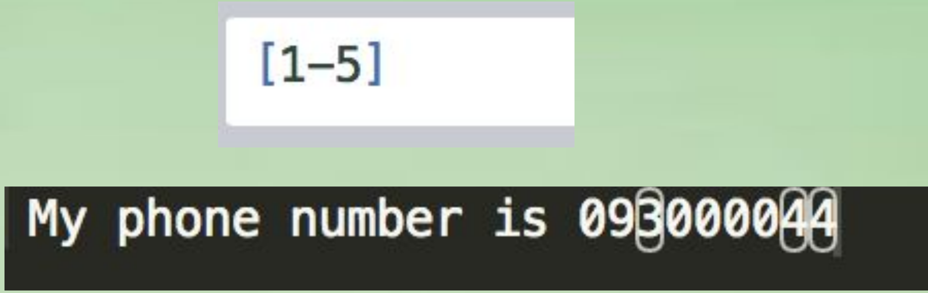
Regular expressions allow specifying ranges of characters with the help of square brackets.

[...] - one of the characters in the brackets
[char1-char2] - one of the characters in the range from char1 to char2.



```
[a-g]
```

```
These are the letters from a to g
```



```
[1-5]
```

```
My phone number is 0930000044
```

Metacharacters

Character Ranges

Regular expressions allow specifying ranges of characters with the help of square brackets.

[...] - one of the characters in the brackets
[char1-char2] - one of the characters in the range from char1 to char2.

Another way to match a word is to write:
[a-z]+
To match digits use **[0-9]**

Metacharacters

Character Ranges

Regular expressions allow specifying ranges of characters with the help of square brackets.

[...] - one of the characters in the brackets
[char1-char2] - one of the characters in the range from char1 to char2.

If you want to use the hyphen as a character in the range, put it at the beginning or at the end of the range

e.g., **[ab-]**

Metacharacters

Character Ranges

Regular expressions allow specifying ranges of characters with the help of square brackets.

[^ ...] - any character except the characters in the brackets. ^ is for negation.
[^char1-char2] - any character except characters in the range from char1 to char2

[^aeuoi] matches one letter that represents a consonant sound, like "b"

Metacharacters

Character Ranges

Regular expressions allow specifying ranges of characters with the help of square brackets.

[^ …] - any character except the characters in the brackets. ^ is for negation.
[^char1-char2] - any character except characters in the range from char1 to char2

If you want to use the caret as a character in the range, don't put it at the beginning of the range

e.g., **[a^b]**

Practice time

Write a regex that matches a word that starts with a vowel and ends with a consonant.

Practice time

Write a regex that matches a word that starts with a vowel and ends with a consonant.

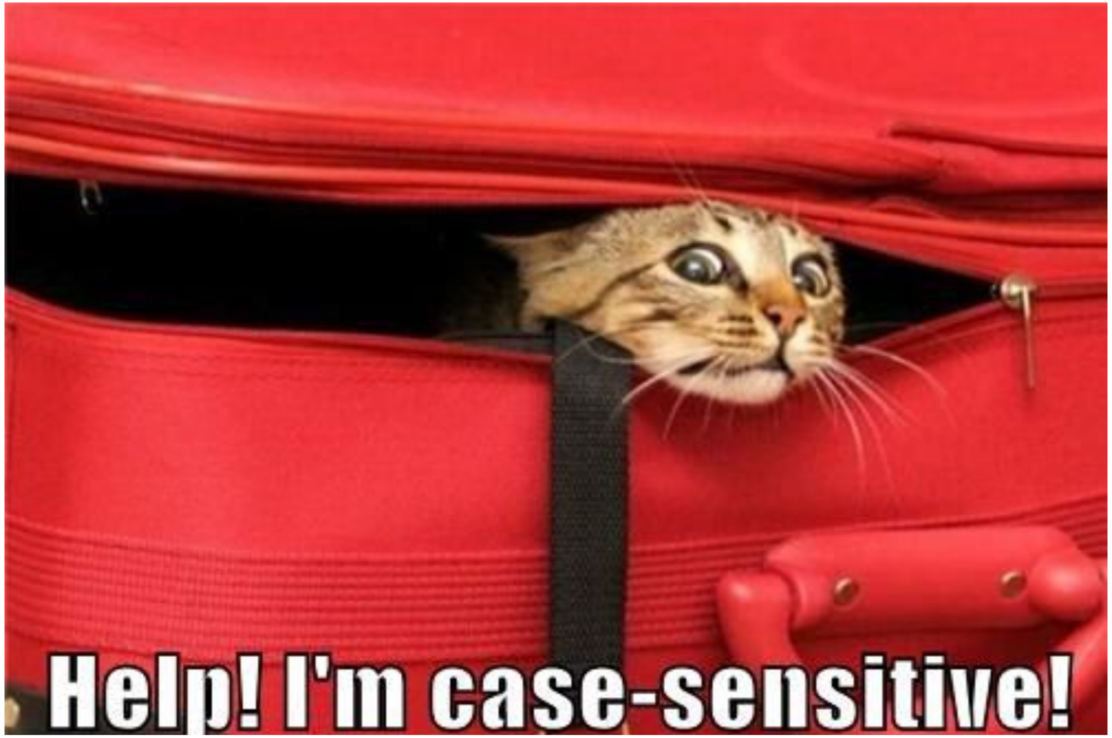
The answer is
\b[ueoia]\w*[^ueoia\s]\b

Logical OR

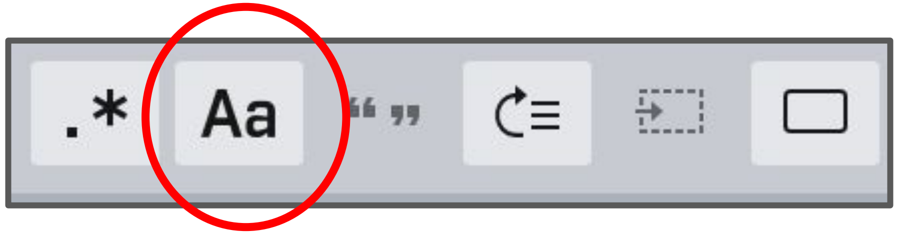
Square brackets allow choosing one character from the specified range. A pipe (|) allows choosing one string from a range.

the (cat|dog|rat)s matches strings "the cats", "the dogs" and "the rats"

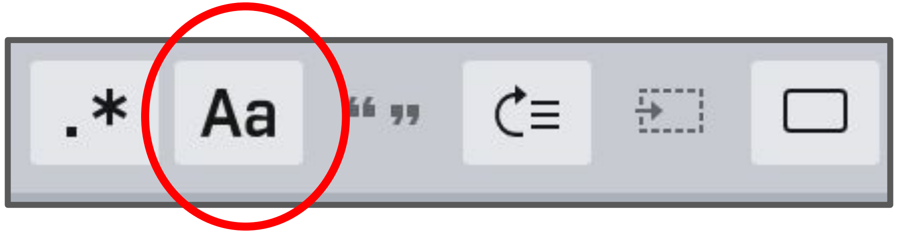
CaSe SEnSiTivity?



In Sublime Text 2, this mode is enabled by default



In Sublime Text 2, this mode is enabled by default



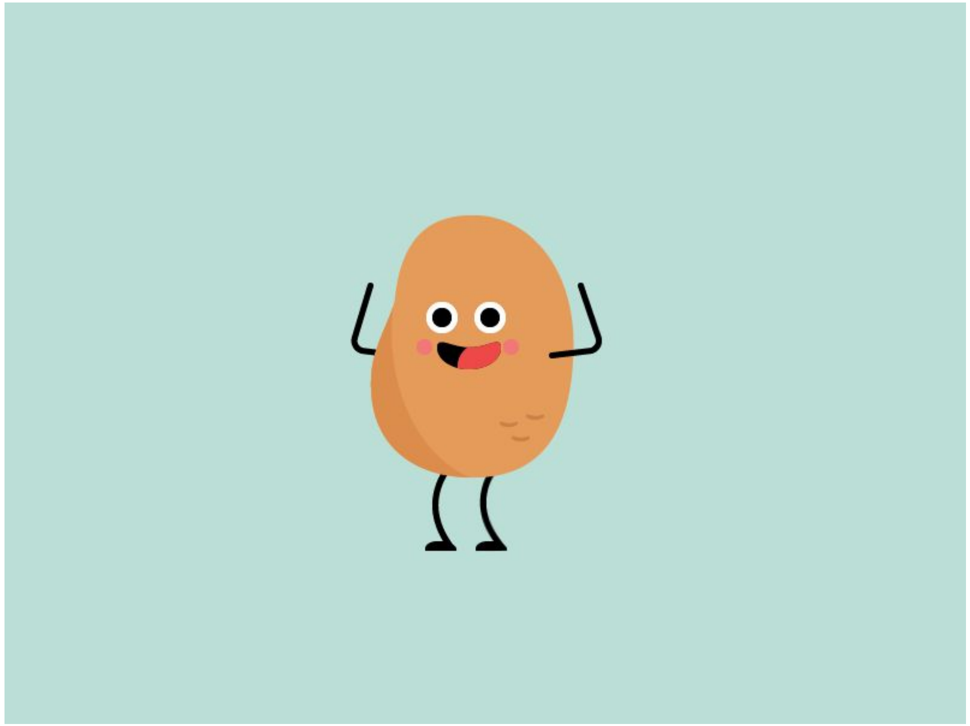
(?i) - case-insensitive mode.

(?i)you matches “You” and “you”

(?i)^bonnie and clyde$ matches “bonnie and clyde” and “Bonnie and Clyde”

It's practice time!

Beginner Level Part 1



Capturing groups

Match group can be referenced later in the expression or used in the replace part.

Capturing groups

Match group can be referenced later in the expression or used in the replace part.

(...) - a match group that can be referenced
\number - a reference to the match group using its position in the regexp

Capturing groups

Match group can be referenced later in the expression or used in the replace part.

(...) - a match group that can be referenced
\number - a reference to the match group using its position in the regexp

\b([a-z]+) [a-z]+ \1 matches " to go to", "as well as", etc.

Capturing groups

Match group can be referenced later in the expression or used in the replace part.

(...) - a match group that can be referenced
\number - a reference to the match group using its position in the regexp

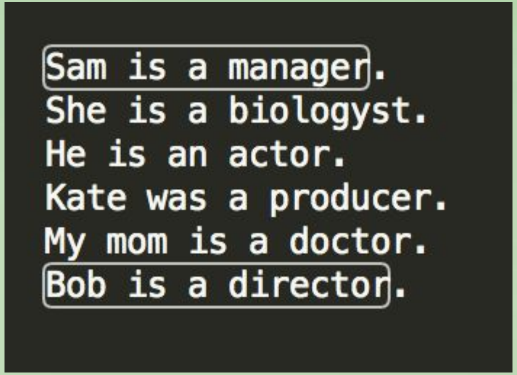
\b([a-z]+) [a-z]+ \1 matches " to go to", "as well as", etc.

(\d+)-(\d+)-\1-\2 matches "11-22-11-22" , “77-72-77-72”, etc

Capturing groups

Match group can be referenced later in the expression or used in the replace part.

(?:...) - a match group that cannot be referenced (a passive group or non-matching group)

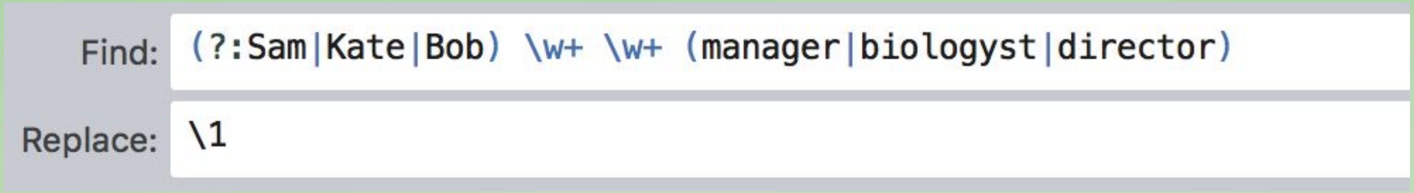


```
Sam is a manager.
She is a biologyst.
He is an actor.
Kate was a producer.
My mom is a doctor.
Bob is a director.
```

Capturing groups

Match group can be referenced later in the expression or used in the replace part.

(?:...) - a match group that cannot be referenced (a passive group or non-matching group)



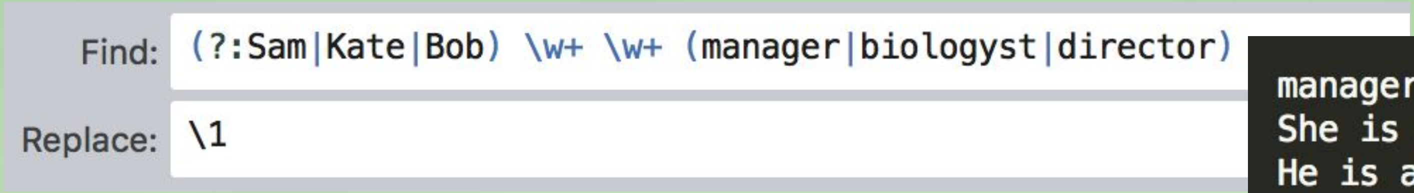
```
(?:Sam|Kate|Bob) \w+ \w+ (manager|biologyst|director)
```

```
\1
```

Capturing groups

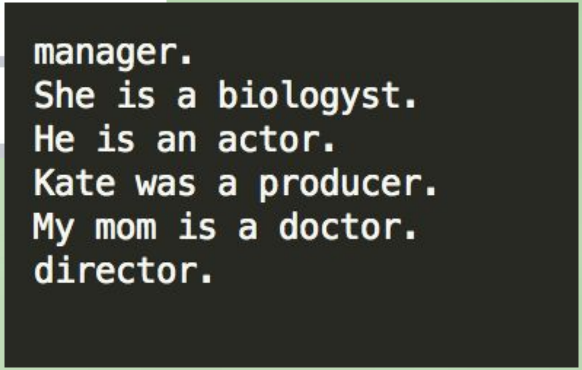
Match group can be referenced later in the expression or used in the replace part.

(?:...) - a match group that cannot be referenced (a passive group or non-matching group)



```
(?:Sam|Kate|Bob) \w+ \w+ (manager|biologyst|director)
```

```
\1
```



```
manager.
She is a biologyst.
He is an actor.
Kate was a producer.
My mom is a doctor.
director.
```

How to match characters, that we use as metacharacters?

Characters like **"."**, **"?"**, **"{"**, etc. have special meaning in the regular expression language.

Matching Literal Characters

In order to match them literally, put a backslash (\) before them.

\. matches a period
\] matches a right square bracket
\$ matches a dollar sign
\\ matches a backslash

It's practice time!

Beginner Level Part 2

Lookarounds

Sometimes you need to look around a bit and see if anything follows or precedes your regexp. This is when the lookaround syntax comes in use.

Lookarounds

Sometimes you need to look around a bit and see if anything follows or precedes your regexp. This is when the lookaround syntax comes in use.

(?=...) - positive lookahead
(?!...) - negative lookahead

Lookarounds

Sometimes you need to look around a bit and see if anything follows or precedes your regexp. This is when the lookaround syntax comes in use.

(?=...) - positive lookahead $\rightarrow$ matches something followed by something else

(?!...) - negative lookahead $\rightarrow$ matches something **not** followed by something else

Lookarounds

Sometimes you need to look around a bit and see if anything follows or precedes your regexp. This is when the lookaround syntax comes in use.

(?=...) - positive lookahead → matches something followed by something else

(?!...) - negative lookahead → matches something **not** followed by something else

iphone(?=\d) matches “iphone” in “iphone6”

e(?!a) matches “e” not followed by “a”

Lookarounds

Sometimes you need to look around a bit and see if anything follows or precedes your regexp. This is when the lookaround syntax comes in use.

(?<=...) - positive lookbehind
(?<!...) - negative lookbehind

(?<=I\s)\w+ matches “know” in “I know”

Lookarounds

You can use any regular expression inside lookahead, but <u>not inside lookbehind.</u>
You can use only plain strings inside lookbehind.
Thus, if you need a few lookbehind assertions at the same place, write them separately

((?<=ipad)|(?<=iphone))\d

matches "6" in "iphone6" and "2" in "ipad2".

Lookarounds

Although lookaheads and lookbehinds are enclosed in round brackets, they do not create match groups.
If you need to reference a regular expression that is inside a lookahead or lookbehind, you should enclose the regexp in another pair of round brackets

(?<=(iphone))\d

has a match group (iphone) that can be referenced later.

Regexp Matching Modes

There are various matching modes available within the regular expression language. They are put before the regexp that should be influenced by this mode.

(?i) - case-insensitive mode.

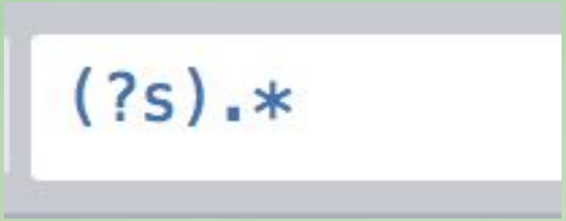
(?s) - DOTALL mode: the period **.** matches newlines, too

Regexp Matching Modes

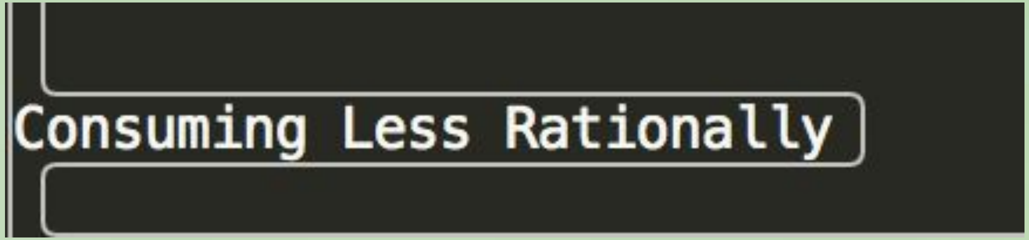
There are various matching modes available within the regular expression language. They are put before the regexp that should be influenced by this mode.

(?i) - case-insensitive mode.

(?s) - DOTALL mode: the period **.** matches newlines, too



```
(?s).*
```



```
Consuming Less Rationally
```

Regexp Matching Modes

There are various matching modes available within the regular expression language. They are put before the regexp that should be influenced by this mode.

(?x) - free-spacing mode: spaces in your regex are ignored. This mode is useful when you want to make your regexp more readable. If you need to use a space in your regexp, you will have to put a slash before it (\ ) or use \s

Regexp Matching Modes

There are various matching modes available within the regular expression language. They are put before the regexp that should be influenced by this mode.

(?x) - free-spacing mode: spaces in your regex are ignored. This mode is useful when you want to make your regexp more readable. If you need to use a space in your regexp, you will have to put a slash before it (\ ) or use \s

```
(?x)
(hello|goodbye)
,?
\s
[A-Z]\w+   #name
```

Use command+enter to go to the new line.

It's practice time!

Medium Level