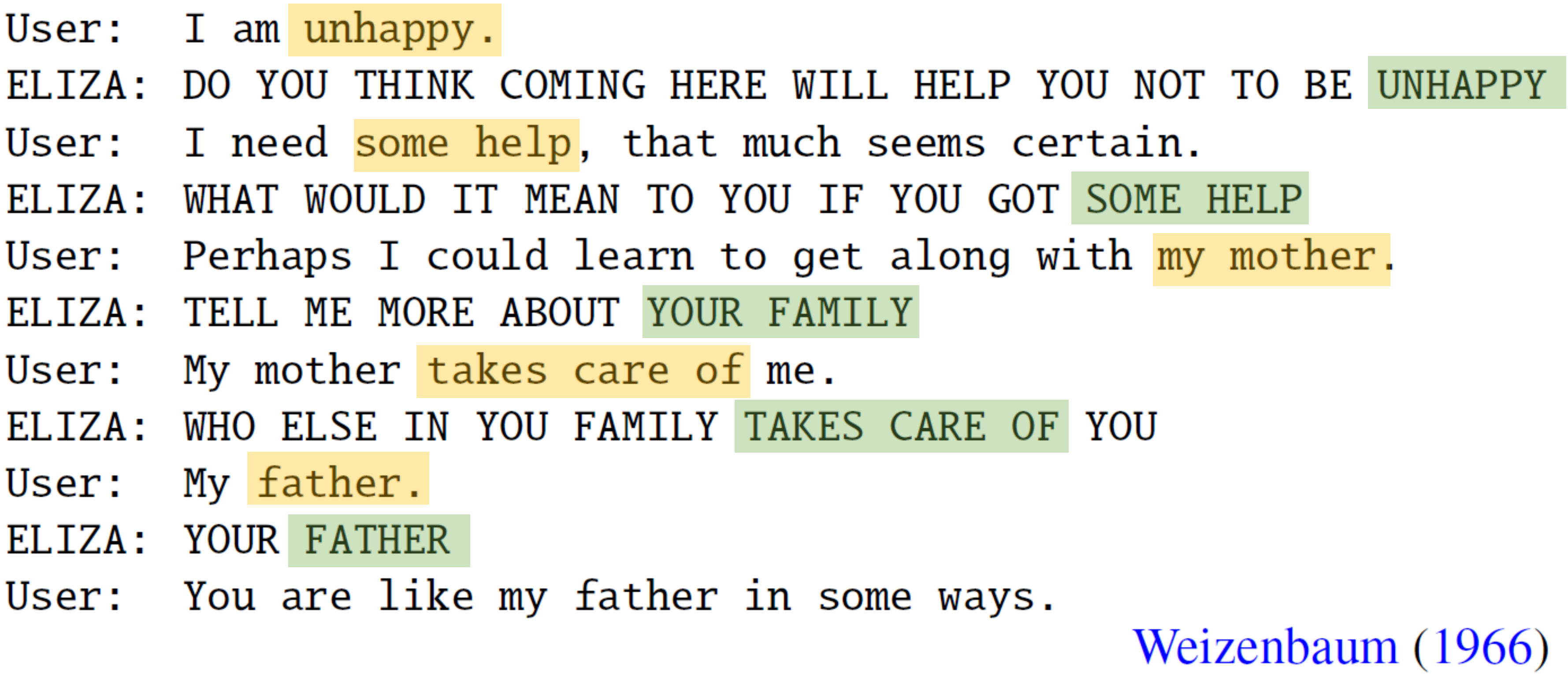
SC4002 CE4045 CZ4045
Natural Language Processing

Regular Expressions

Dr. Sun Aixin

Chatbot

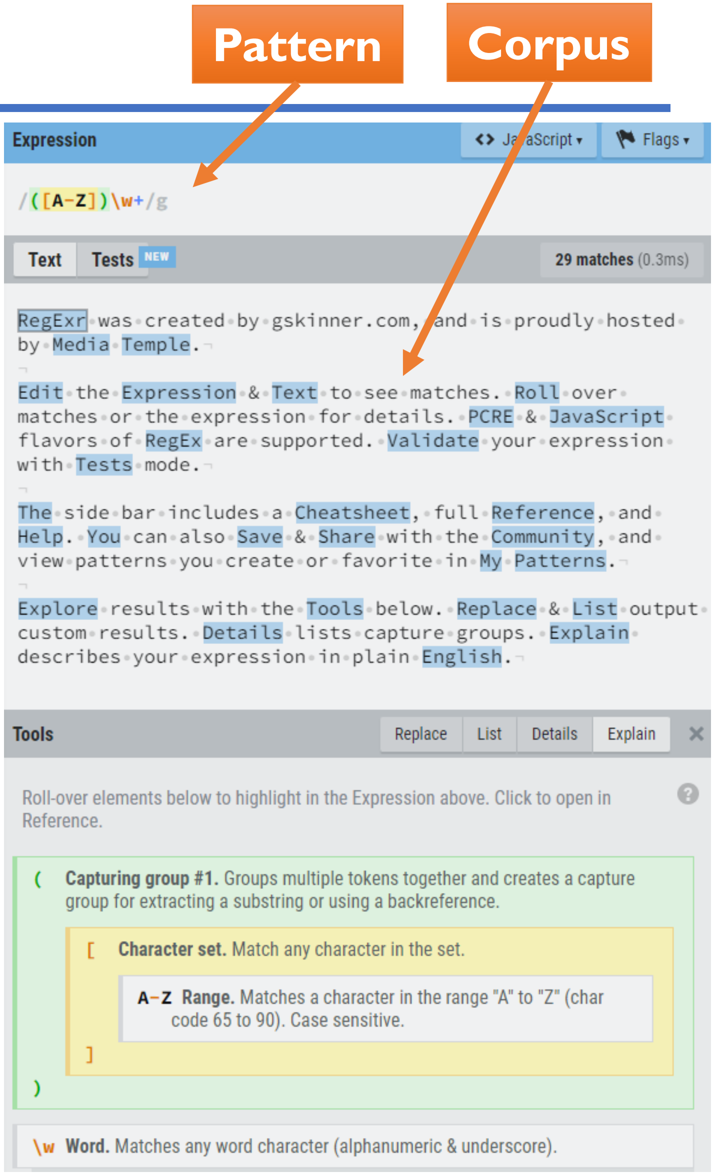


User: I am unhappy.
ELIZA: DO YOU THINK COMING HERE WILL HELP YOU NOT TO BE UNHAPPY
User: I need some help, that much seems certain.
ELIZA: WHAT WOULD IT MEAN TO YOU IF YOU GOT SOME HELP
User: Perhaps I could learn to get along with my mother.
ELIZA: TELL ME MORE ABOUT YOUR FAMILY
User: My mother takes care of me.
ELIZA: WHO ELSE IN YOU FAMILY TAKES CARE OF YOU
User: My father.
ELIZA: YOUR FATHER
User: You are like my father in some ways.

Weizenbaum (1966)

Regular Expressions (RE)

- RE is a language for specifying text search strings.
  - Used in every computer language, word processor, and text processing tools
  - Formally, a RE is an **algebraic notation** for characterizing a **set** of strings.
- Particularly useful for searching in texts, when we have a **pattern** to search for and a **corpus** of texts to search through.
  - A RE search function will either return the first match, or search through the corpus and return all texts that match the pattern
  - The corpus can be a **single document** or a **collection**
  - Example: search for postcode in 5 documents



Pattern

Corpus

```
/([A-Z])\w+/g
```

29 matches (0.3ms)

```
RegExr was created by gskinner.com, and is proudly hosted by Media Temple.

Edit the Expression & Text to see matches. Roll over matches or the expression for details. PCRE & JavaScript flavors of RegEx are supported. Validate your expression with Tests mode.

The side bar includes a Cheatsheet, full Reference, and Help. You can also Save & Share with the Community, and view patterns you create or favorite in My Patterns.

Explore results with the Tools below. Replace & List output custom results. Details lists capture groups. Explain describes your expression in plain English.
```

Roll-over elements below to highlight in the Expression above. Click to open in Reference.

`(` **Capturing group #1.** Groups multiple tokens together and creates a capture group for extracting a substring or using a backreference.

`[` **Character set.** Match any character in the set.

`A-Z` **Range.** Matches a character in the range "A" to "Z" (char code 65 to 90). Case sensitive.

`]`

`)`

`\w` **Word.** Matches any word character (alphanumeric & underscore).

Basic Regular Expression Patterns

- The simplest RE is a sequence of simple characters, like /test/
  - Putting characters in sequence is called **concatenation**.
  - / / is not part of the RE, but the notation to indicate RE.
  - /test/ will match test in text, but not match Test, or TEST.
  - Regular expressions are **case sensitive.**

RE	Example patterns matched
/woodchucks/	"interesting links to woodchucks and lemurs"
/a/	"Mary Ann stopped by Mona's"
/!/	"You've left the burglar behind again!" said Nori

- In our examples, we generally underline the first exact part of the match.
  - If the word woodchucks appears to be the first word in a sentence, then *Woodchucks* will not be match.

Disjunction of characters to match

- The string of characters inside the square braces [ ] specifies a disjunction of characters to match
  - Square brackets [ ] matches any single character from within the class

RE	Match	Example Patterns
/[wW]oodchuck/	Woodchuck or woodchuck	"Woodchuck"
/[abc]/	'a', 'b', or 'c'	"In uomini, in soldati"
/[1234567890]/	any single digit	"plenty of 7 to 5"

- If there is a well-defined sequence associated with a set of characters, [ ] can be used with the dash (-) to specify any one character in a range.
  - /[A-Z]/ matches an upper-case letter
  - /[a-z]/ matches a lower-case letter
  - /[0-9]/ matches a single digit

More on square braces in RE

- The square braces can also be used to specify what a single character cannot be matched with caret ^.
  - If the caret ^ is **the first symbol after the open square brace** [, the resulting pattern is negated.
  - Pattern /[^a]/ matches any single character (including special characters) except a.
  - This is only true when the caret is the first symbol after the open square brace. If it occurs anywhere else, it usually stands for a caret

RE	Match (single characters)	Example Patterns Matched
/[^A-Z]/	not an upper-case letter	"O**y**fn pripetchik"
/[^Ss]/	neither 'S' nor 's'	"**I** have no exquisite reason for't"
/[^.]/	not a period	"**o**ur resident Djinn"
/[e^]/	either 'e' or '^'	"look up **^** now"
/a^b/	the pattern 'a^b'	"look up **a^b** now"

Counters in RE, and wildcard expression

- Question mark ? matches zero or one appearance of the preceding item
- Kleene * (generally pronounced "cleany star") matches zero or more occurrences of the immediately previous character or regular expression
- Kleene + matches one or more occurrences of the immediately preceding character or regular expression.
- Examples
  - /colou?r/ matches color or colour
  - /ba*!/ matches b!, ba!, baaa!, baaaaa! and more such strings
  - /ba+!/ matches ba!, baa!, baaaaa! and more such strings
  - /[ab]+/ means "one or more *a*'s or *b*'s" like aaaa, or ababbb or bbbb
- The period (/./) is a wildcard expression that matches any single character (except a carriage return), e.g., /beg.n/ matches begin, beg'n, begun
  - The wildcard used together with the Kleene star /.*/ means "any string of characters".
  - When the period is used inside a negation /[^.]/ , it means period and not the wildcard expression.

Anchors in RE

- Anchors are special characters that anchor regular expressions to particular places in a string
- The caret **^** matches the start of a line.
  - The pattern /^The/ matches the word The only at the start of a line.
- The dollar sign **$** matches the end of a line.
  - /^The dog\.$/ matches a line that contains only the phrase "The dog."
  - Backslash dot \. is escaped character, means the full stop or period, not the wildcard.
- Two other anchors: word boundary \b and non-word boundary \B
  - A "word" for the purposes of a regular expression is defined as any sequence of digits, underscores, or letters, based on the definition of "words" in programming languages.
  - /\bthe\b/ matches the word the but not the word other
  - /\b99\b/ will match "There are 99 bottles" (because 99 follows a space) but not "There are 299 bottles" (since 99 follows a number). But it will match $99, since 99 follows a dollar sign (not part of the word definition above).

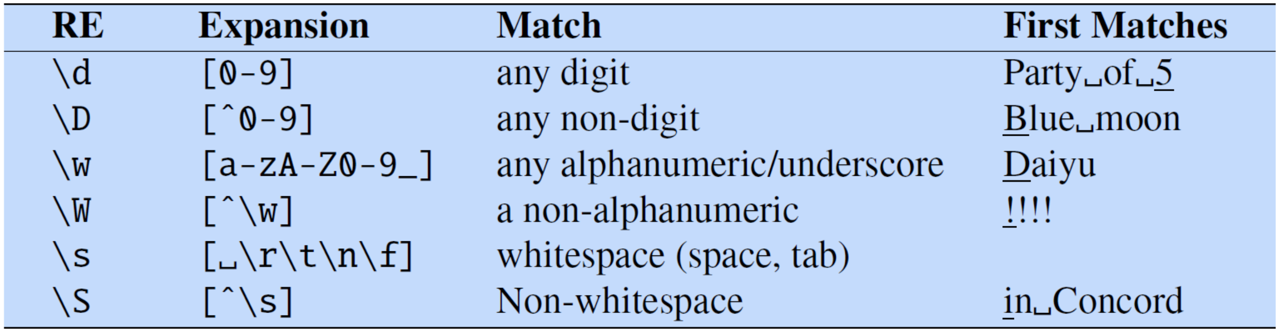
Disjunction, Grouping, and Precedence

- The disjunction operator, also called the pipe symbol |, means "or".
  - The pattern /cat|dog/ matches either the string cat or the string dog
- Parenthesis operators (and).
  - Enclosing a pattern in parentheses makes it act like a single character for the purposes of neighboring operators like the pipe | and the Kleene*.
  - /gupp(y|ies)/ would specify that we meant the disjunction only to apply to the suffixes y and ies. The RE matches guppy or guppies
  - /(abb)+/ matches abb, abbabb, abbabbabb ….

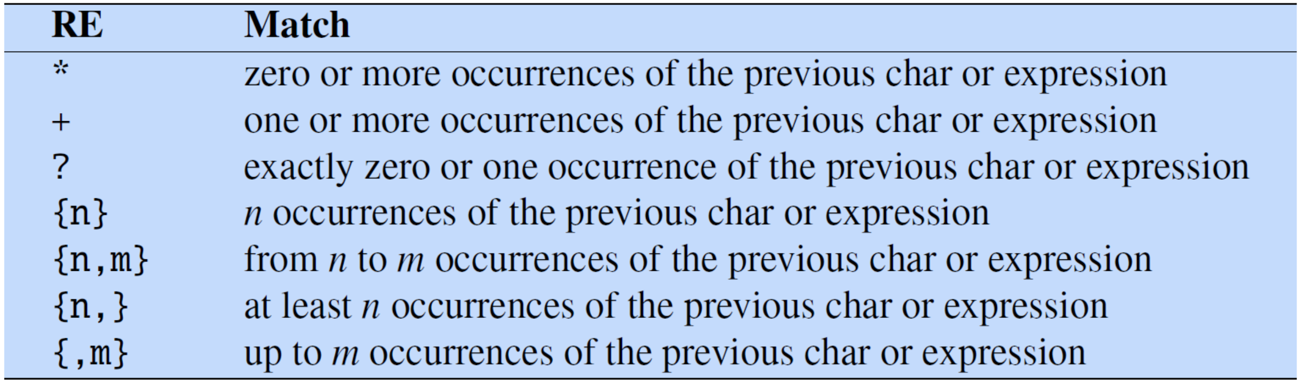
REs always match the largest string they can!

- Operator precedence
  - Parenthesis ( )
  - Counters * + ? {}
  - Sequences and anchors the ^my end$
  - Disjunction |

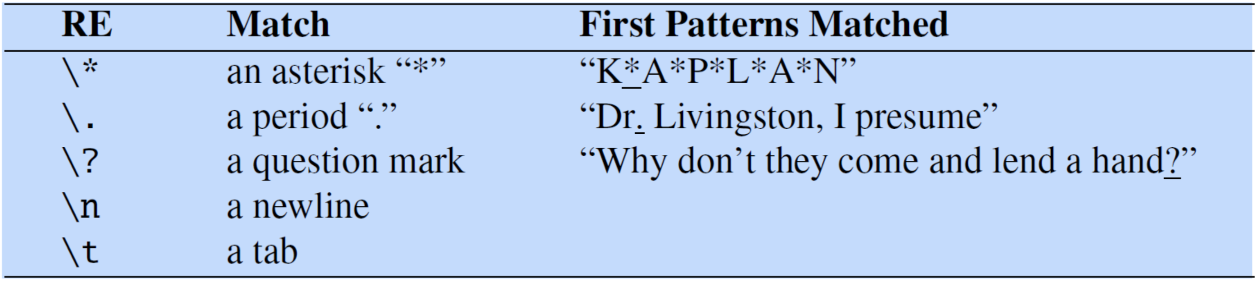
More operators



RE	Expansion	Match	First Matches
\d	[0-9]	any digit	Party␣of␣5
\D	[^0-9]	any non-digit	Blue␣moon
\w	[a-zA-Z0-9_]	any alphanumeric/underscore	Daiyu
\W	[^\w]	a non-alphanumeric	!!!!
\s	[␣\r\t\n\f]	whitespace (space, tab)	
\S	[^\s]	Non-whitespace	in␣Concord



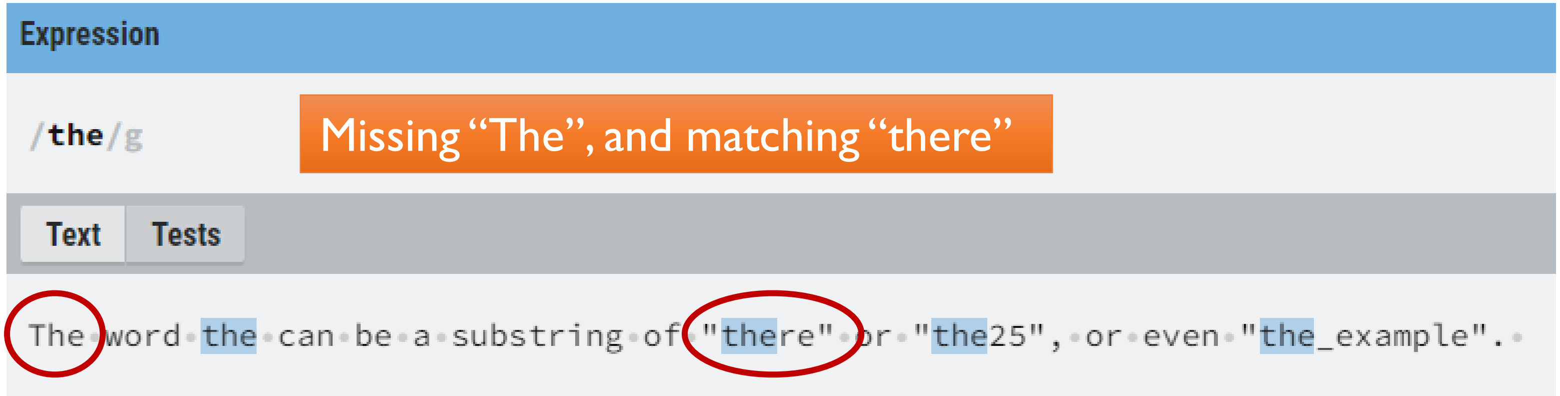
RE	Match
*	zero or more occurrences of the previous char or expression
+	one or more occurrences of the previous char or expression
?	exactly zero or one occurrence of the previous char or expression
{n}	*n* occurrences of the previous char or expression
{n,m}	from *n* to *m* occurrences of the previous char or expression
{n,}	at least *n* occurrences of the previous char or expression
{,m}	up to *m* occurrences of the previous char or expression



RE	Match	First Patterns Matched
*	an asterisk "*"	"K*A*P*L*A*N"
\.	a period "."	"Dr. Livingston, I presume"
\?	a question mark	"Why don't they come and lend a hand?"
\n	a newline	
\t	a tab	

A working example

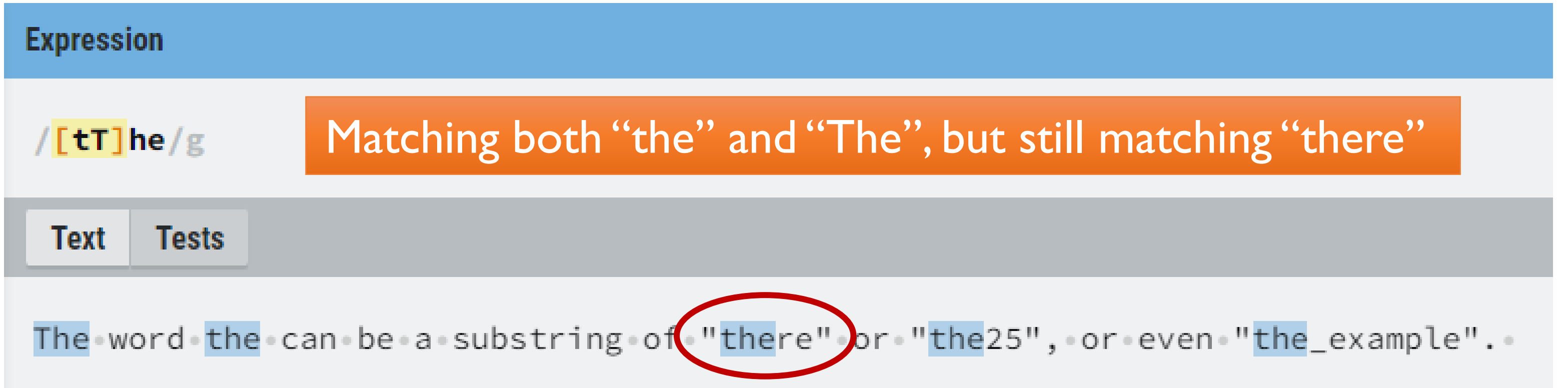
- Suppose we wanted to write a RE to find the English article *the* in text.



`/the/g`

Missing "The", and matching "there"

`The word the can be a substring of "there" or "the25", or even "the_example".`



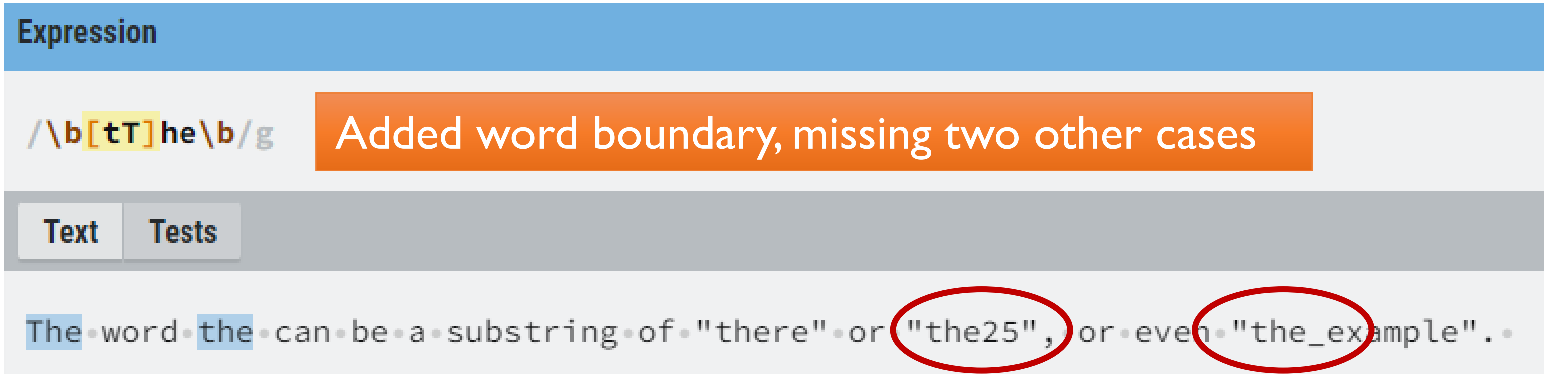
`/[tT]he/g`

Matching both "the" and "The", but still matching "there"

`The word the can be a substring of "there" or "the25", or even "the_example".`

A working example

- Suppose we wanted to write a RE to find the English article *the* in text.

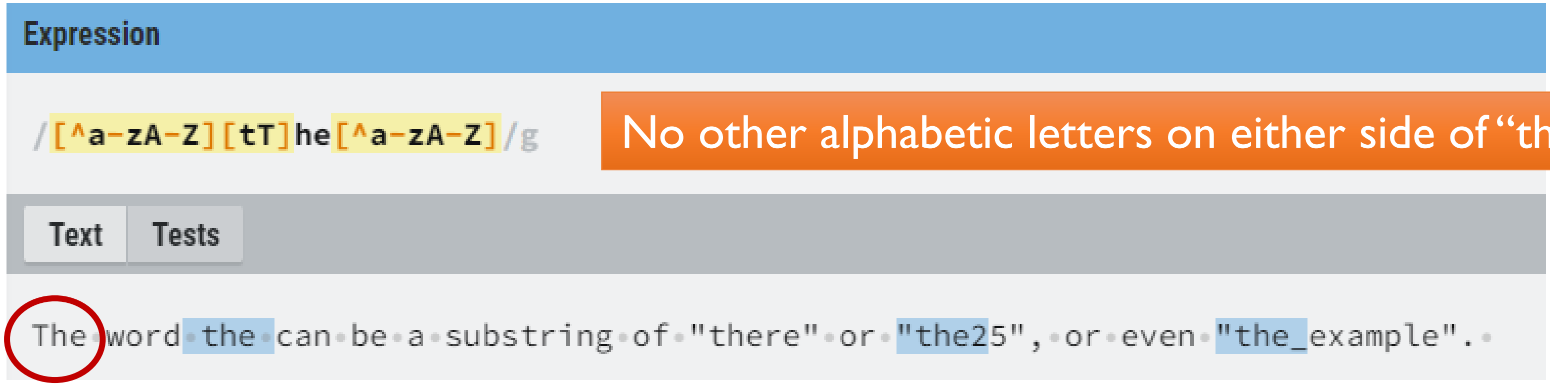


Expression

`/\b[tT]he\b/g`

Added word boundary, missing two other cases

The word the can be a substring of "there" or "the25", or even "the_example".



Expression

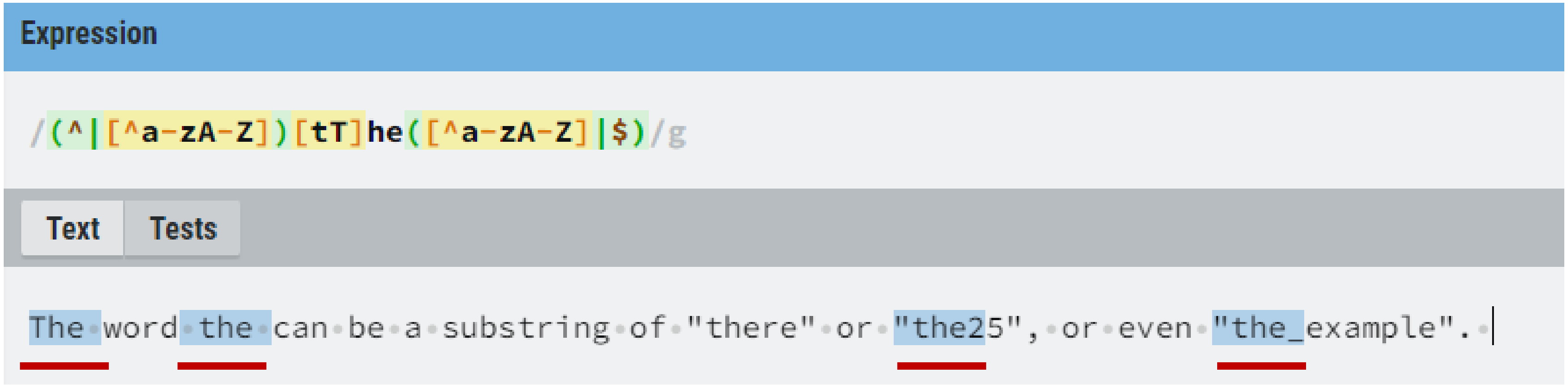
`/[^a-zA-Z][tT]he[^a-zA-Z]/g`

No other alphabetic letters on either side of “the”

The word the can be a substring of "there" or "the25", or even "the_example".

A working example

- Suppose we wanted to write a RE to find the English article *the* in text.



```
/(^|[^a-zA-Z])[tT]he([^a-zA-Z]|$)/g
```

```
The word the can be a substring of "there" or "the25", or even "the_example".
```

  - A more complicated (but not perfect) version
  - Match "the" or "The" either in text or at the beginning of the line, or at the end of line.
  - There is no alphabetic letters on either side of the word "the".
  - The final version can be application dependent.

False Positive and False Negative

- The whole process is to fix two kinds of errors:
  - **False positives**: strings that we incorrectly matched like other or there,
  - **False negatives**: strings that we incorrectly missed, like The.

- Addressing these two kinds of errors is common in implementing language processing systems.
- Reducing the overall error rate for an application thus involves
  - Increasing **precision** (minimizing false positives)
  - Increasing **recall** (minimizing false negatives)

Summary

- RegExr: an online tool to learn, build, & test Regular Expressions
  - http://regexr.com/

- Java RegEx API and Tutorial
  - http://docs.oracle.com/javase/8/docs/api/java/util/regex/package-summary.html
  - http://docs.oracle.com/javase/tutorial/essential/regex/

- Reference: https://web.stanford.edu/~jurafsky/slp3/
  - **Chapter 2**, Regular Expressions, Text Normalization, Edit Distance

What can we do?

- Given a document, we are able to search for the matching strings with a query specified in Regular Expression
  - The given document is basically a sequence of characters
  - At this stage, we do not understand words or sentences in the document.

- Next, it would be useful to recognize the words, sentences in the document
  - With the words and sentences, we will be able to understand the structure or meaning of the sentences.