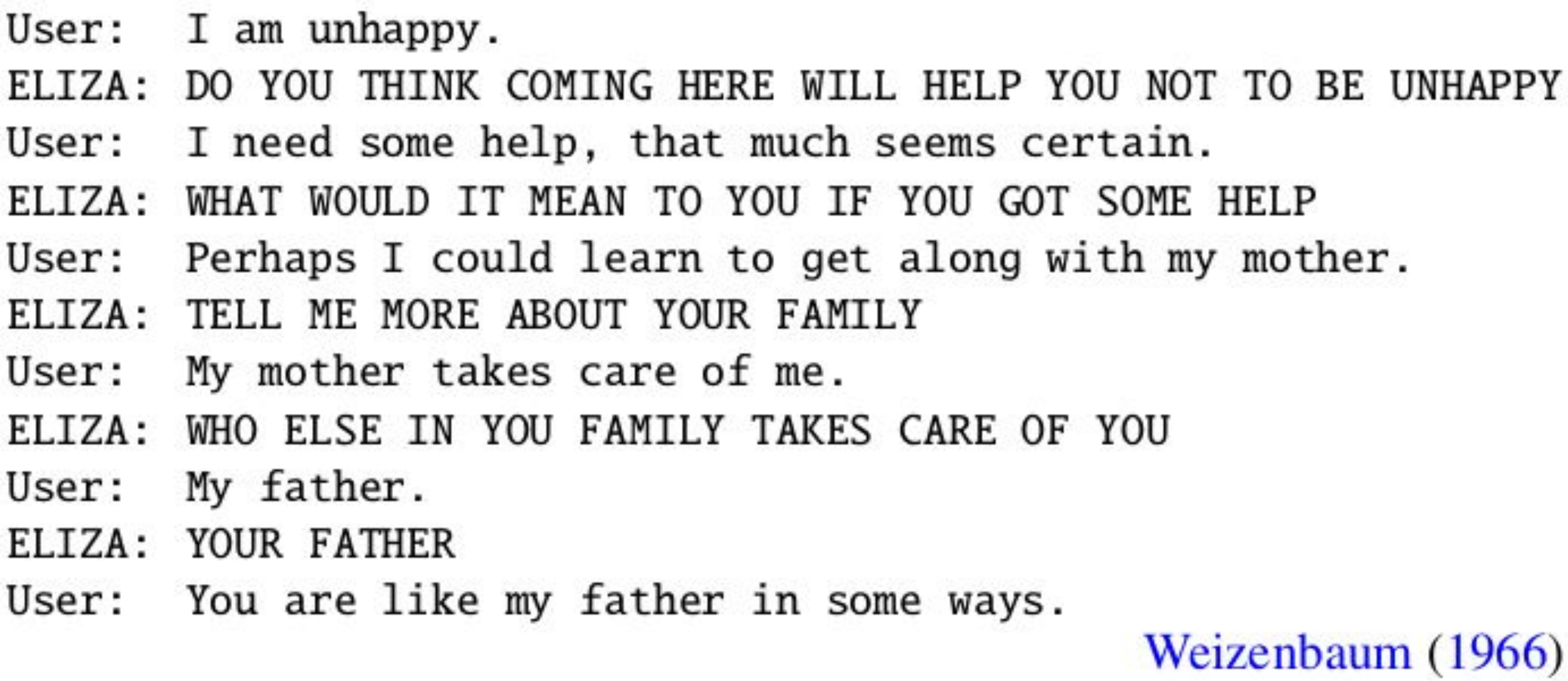
COM1013
INTRODUCTION TO COMPUTER SCIENCE

Lecturer: Begüm MUTLU BİLGE, PhD

begummutlubilge+com1013@gmail.com (recommended)
bmbilge@ankara.edu.tr

ELIZA



```
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)

Text Normalization

Normalizing text means converting it to a more convenient, standard form.

- Regular expressions play an important part
    - Tokenization
    - Lemmatization
    - Stemming
    - Sentence segmentation

Regular Expressions

A language for specifying text search strings

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.

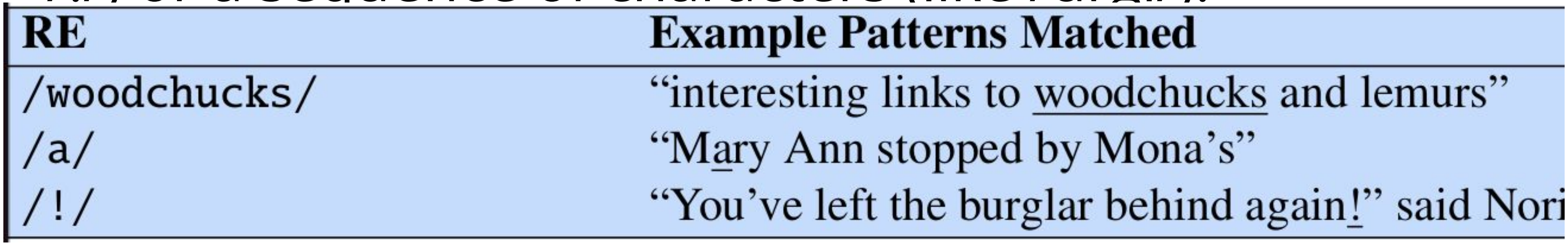
A regular expression search function will search through the corpus, returning all texts that match the pattern.

Basic Regular Expression Patterns

The simplest kind of regular expression is a sequence of simple characters.

- To search for woodchuck, we type /woodchuck/.
- The expression /Buttercup/ matches any string containing the substring Buttercup; grep with that expression would return the line “I’m called little Buttercup.”

The search string can consist of a single character (like /!/) or a sequence of characters (like /urgl/).



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

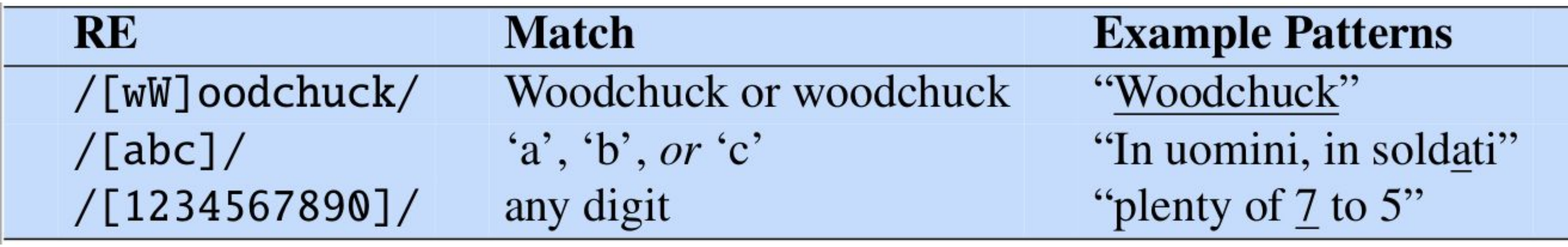
Basic Regular Expression Patterns

Regular expressions are case sensitive;

- Lower case /s/ is distinct from uppercase /S/ (/s/ matches a lower case s but not an uppercase S)
- The pattern /**w**oodchucks/ will not match the string **W**oodchucks.

Solution : Braces [and]

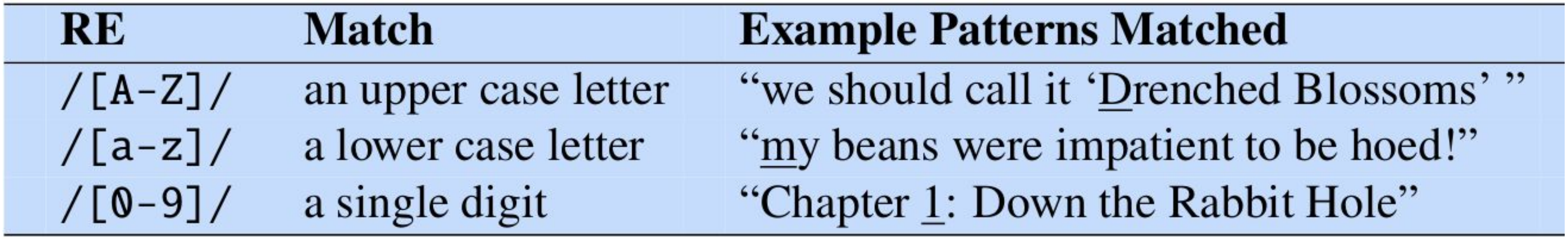
- Braces specifies a **disjunction** of characters to match



RE	Match	Example Patterns
`/[wW]oodchuck/`	Woodchuck or woodchuck	“Woodchuck”
`/[abc]/`	‘a’, ‘b’, *or* ‘c’	“In uomini, in soldati”
`/[1234567890]/`	any digit	“plenty of 7 to 5”

Basic Regular Expression Patterns

The brackets can be used with the dash (-) to specify any one character in a **range**.

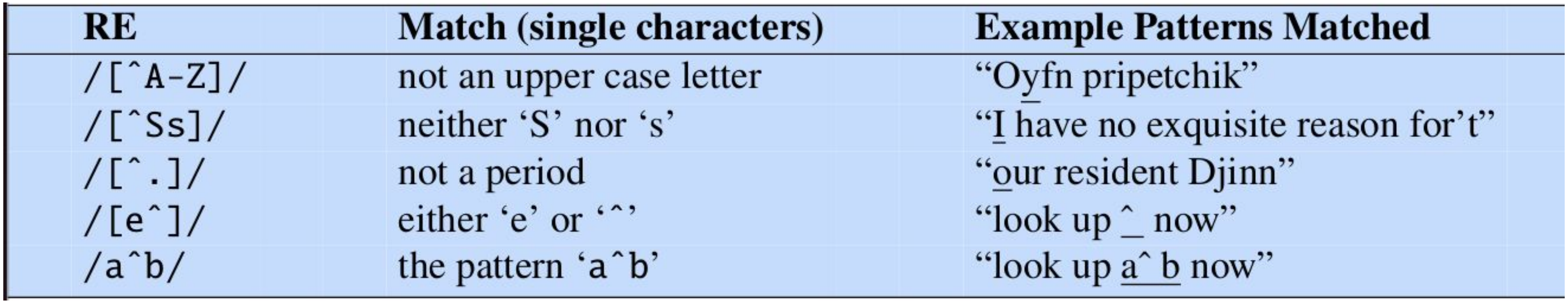


RE	Match	Example Patterns Matched
`/[A-Z]/`	an upper case letter	“we should call it ‘Drenched Blossoms’ ”
`/[a-z]/`	a lower case letter	“my beans were impatient to be hoed!”
`/[0-9]/`	a single digit	“Chapter 1: Down the Rabbit Hole”

Basic Regular Expression Patterns

The square braces can also be used to specify what a single character cannot be, by use of the caret ˆ.

- If the caret ˆ is the **first** symbol after the open square brace [, the resulting pattern is **negated**.
- If the caret ˆ occurs **anywhere else**, it usually stands for a **regular** caret character.

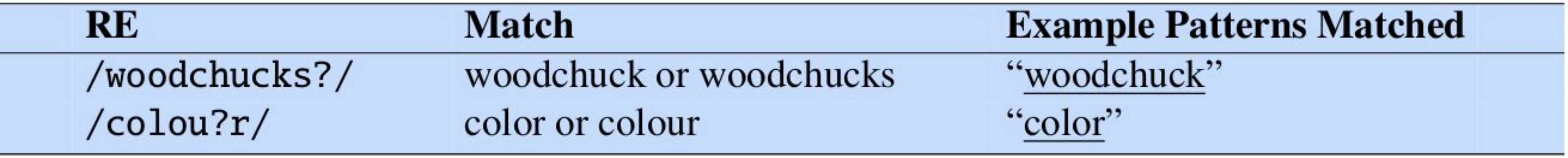


RE	Match (single characters)	Example Patterns Matched
`/[ˆA-Z]/`	not an upper case letter	“Oyfn pripetchik”
`/[ˆSs]/`	neither ‘S’ nor ‘s’	“I have no exquisite reason for’t”
`/[ˆ.]/`	not a period	“our resident Djinn”
`/[eˆ]/`	either ‘e’ or ‘ˆ’	“look up ˆ now”
`/aˆb/`	the pattern ‘aˆb’	“look up aˆ b now”

Basic Regular Expression Patterns

Optional elements

- The question mark /?/, which means “the preceding character or nothing”
    - “zero or one instances of the previous character”



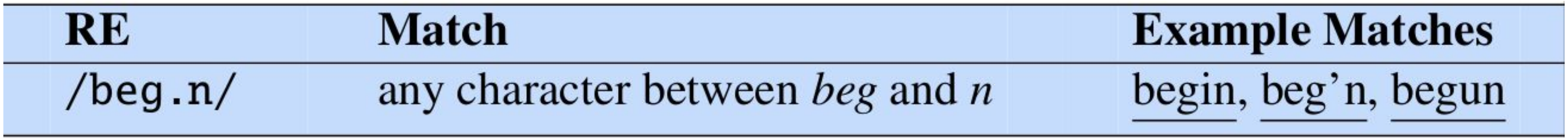
RE	Match	Example Patterns Matched
`/woodchucks?/`	woodchuck or woodchucks	“woodchuck”
`/colou?r/`	color or colour	“color”

- The Kleene star * means “zero or more occurrences of the immediately previous character or regular expression”.
- Kleene +, means “one or more occurrences of the immediately preceding character or regular expression”

RE	Match	RE	Match
/ba*/	b, ba, baa, baaa …	/(ba)+/	ba, baba, bababa …
/ba+/	ba, baa, baaa, baaaa …	/(b|a)+/	b, a, bb, aa, ba, abb, aabba …
/(ba)*/	ε, ba, baba, bababa …	/[0-9]+/	a sequence of digits

Basic Regular Expression Patterns

The period (/./), a wildcard expression that matches any single character (except a carriage return).



RE	Match	Example Matches
`/beg.n/`	any character between *beg* and *n*	begin, beg'n, begun

The wildcard is often used together with the Kleene star to mean “any string of characters”

- E.g. suppose we want to find any line in which a particular word, for example, aardvark, appears twice.
- We can specify this with the regular expression **/aardvark.*aardvark/**.

Basic Regular Expression Patterns

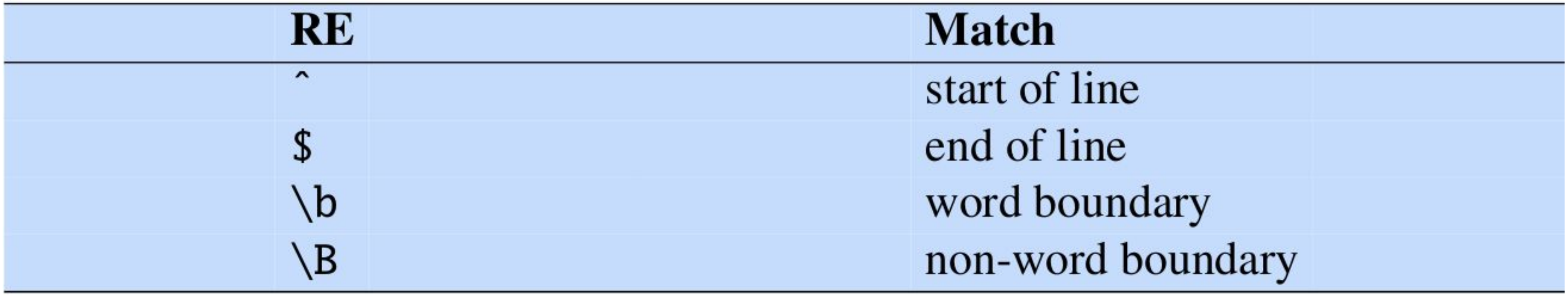
Anchors are special characters that anchor regular expressions to particular places in a string.

- Caret ^
    - to match the start of a line,
    - to indicate a negation inside of square brackets [], and
    - just to mean a caret.
- The dollar sign $
    - matches the end of a line.

/ˆThe dog\.$/ matches a line that contains only the phrase **The dog.**

Basic Regular Expression Patterns

Anchors are special characters that anchor regular expressions to particular places in a string.



RE	Match
^	start of line
$	end of line
\b	word boundary
\B	non-word boundary

`/\bthe\b/` matches the word **the** but not the word **other**.

Disjunction, Grouping, Precedence

- The disjunction operator, also called the pipe symbol |.
    - The pattern /cat|dog/ matches either the string cat or the string dog.
- Enclosing a pattern in parentheses () makes it act like a single character for the purposes of neighboring operators

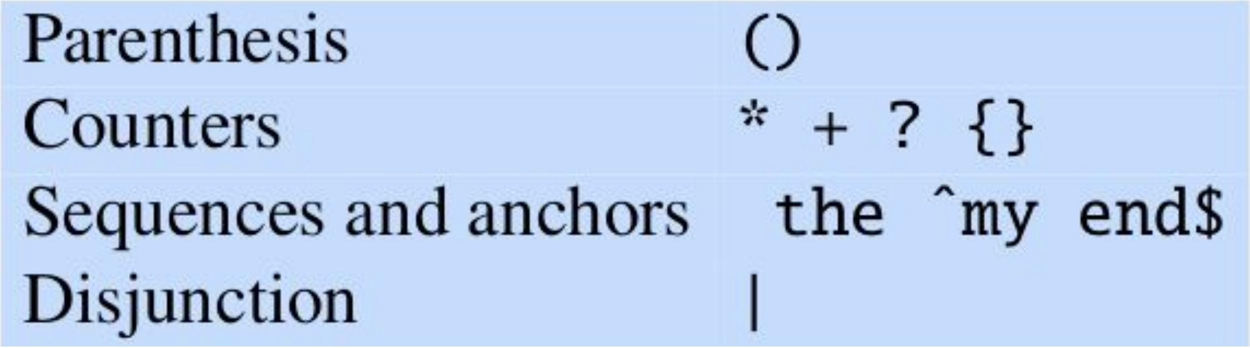
Disjunction, Grouping, Precedence

E.g. Perhaps we have a line that has column labels of the form **Column 1 Column 2 Column 3**.

- The expression **/Column [0-9]+ */** will not match any number of columns; instead, it will match a single column followed by any number of spaces!
- With the parentheses, we could write the expression **/(Column [0-9]+ *)*/** to match the word Column, followed by a number and optional spaces, the whole pattern repeated zero or more times.

Disjunction, Grouping, Precedence

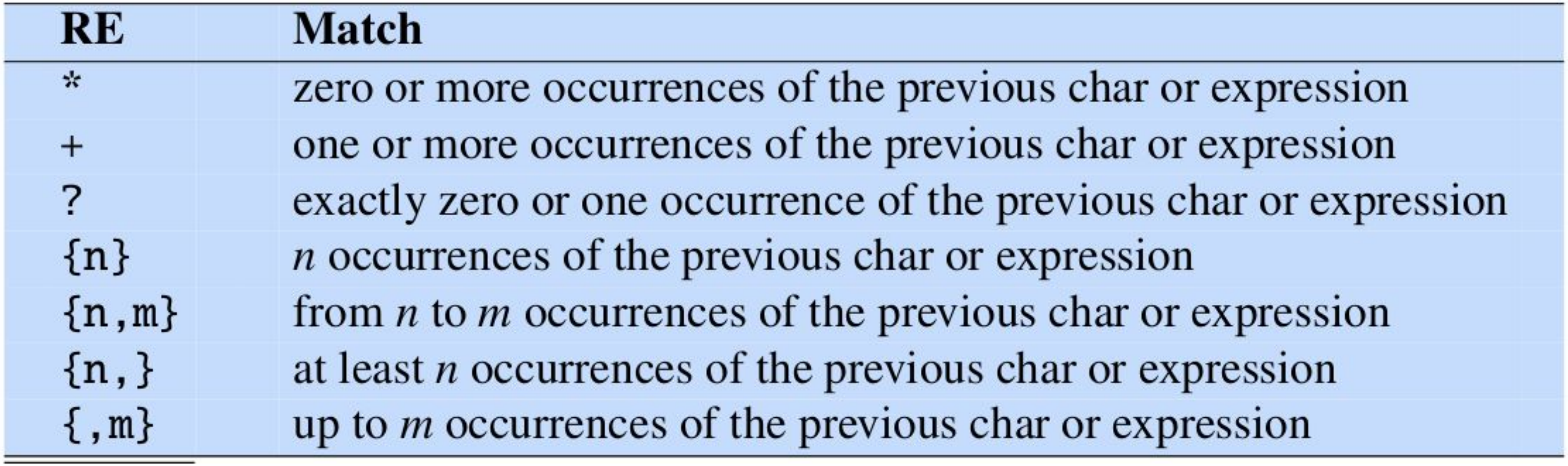
The precedence of operators



Parenthesis	()
Counters	* + ? {}
Sequences and anchors	the ˆmy end$
Disjunction	|

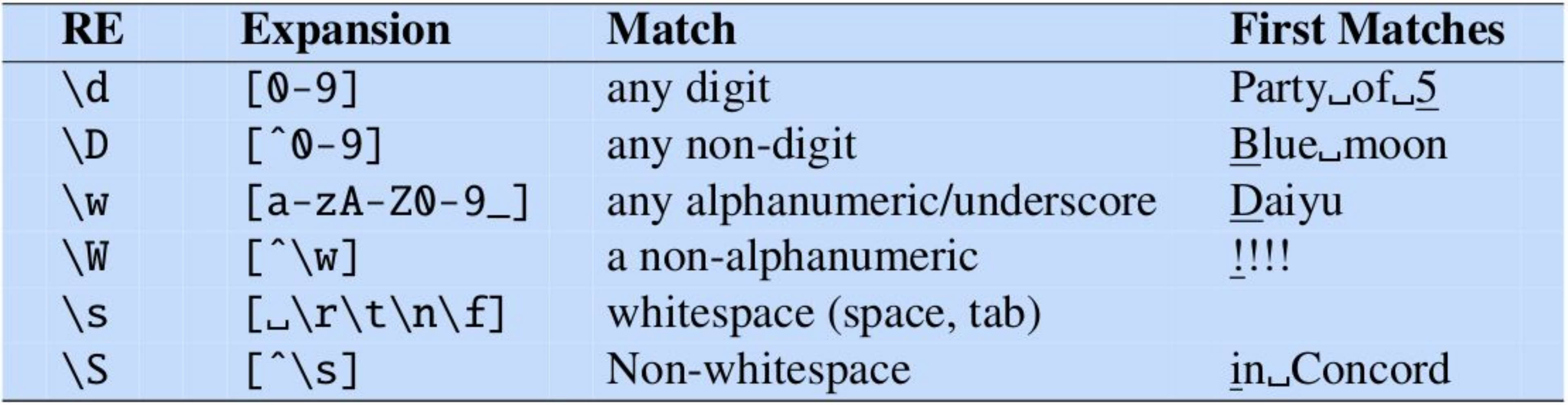
Regular expression operators for counting.

E.g. The RE /a\.{24}z/ will match **a** followed by **24 dots** followed by **z**



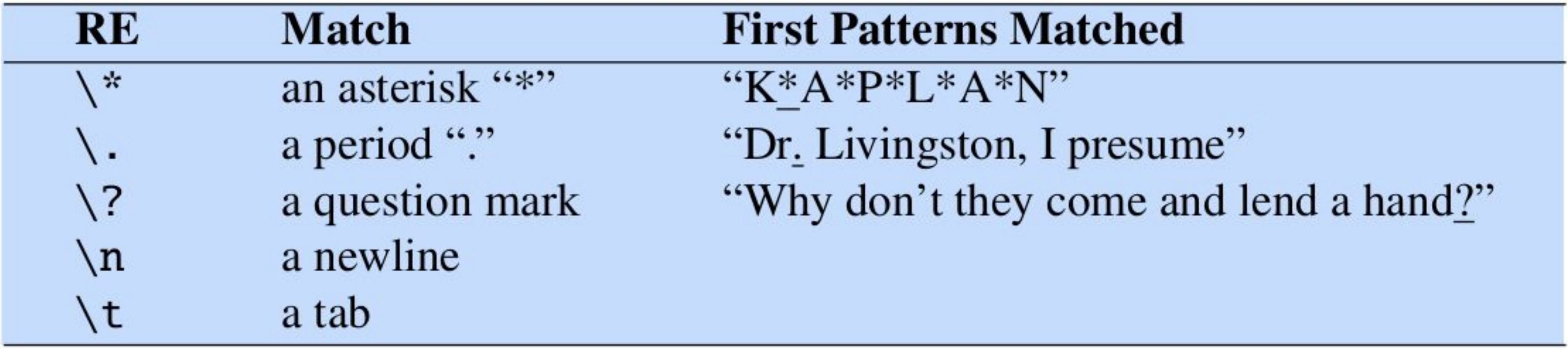
RE	Match
*	zero or more occurrences of the previous char or expression
+	one or more occurrences of the previous char or expression
?	exactly zero or one occurrence of the previous char or expression
{n}	*n* occurrences of the previous char or expression
{n,m}	from *n* to *m* occurrences of the previous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

Aliases for common sets of characters



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

Some characters that need to be backslashed



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	

Example I

Suppose we wanted to write a RE to find cases of the English article *the*.

➔ /the/
➔ /[tT]he/
➔ /\b[tT]he\b/
➔ /[^a-zA-Z][tT]he[^a-zA-Z]/
❖ /(^|[^a-zA-Z])[tT]he([^a-zA-Z]|$)/

Evaluation

The process we just went through was based on fixing two kinds of errors:

- Matching strings that we should not have matched (there, then, other)

  False positives (Type I errors)

- Not matching things that we should have matched (The)

  False negatives (Type II errors)

Evaluation cont.

In NLP we are always dealing with these kinds of errors.

Reducing the error rate for an application often involves two antagonistic efforts:

- Increasing accuracy or precision (minimizing false positives)
- Increasing coverage or recall (minimizing false negatives).

Example II

Suppose we want to build an application to help a user buy a computer on the Web. The user might want “**any machine with at least 6 GHz and 500 GB of disk space for less than $1000**”.

- Regular expression for prices.
    - `/$[0-9]+/`
    - `/$[0-9]+\.[0-9][0-9]/`
    - `/(^|\W)$[0-9]+(\.[0-9][0-9])?\b/`
    - `/(^|\W)$[0-9]{0,3}(\.[0-9][0-9])?\b/`

Substitution, Capture Groups, and ELIZA

Substitutions → s/regexp1/pattern/

- E.g. suppose we wanted to put angle brackets around all integers in a text,
    - For example, changing **the 35 boxes** to **the <35> boxes**.
- Put parentheses (and) around the first pattern and use the **number operator** \1 in the second pattern to refer back.
    - Here's how it looks: s/**(**[0-9]+**)**/<**\1**>/

Substitution, Capture Groups, and ELIZA

Capture Grouping →the use of **parentheses** to **store a pattern** in memory

The parenthesis and number operators can also specify that a certain string or expression must occur twice in the text.

Substitution, Capture Groups, and ELIZA

Capture Grouping

- E.g. Suppose we are looking for the pattern "the **X**er they were, the **X**er they will be", where we want to constrain the two X's to be the same string.
- We do this by surrounding the first **X** with the parenthesis operator, and replacing the second **X** with the number operator **\1**.
  - /the **(.*)**er they were, the **\1**er they will be/
  - **Example match:** the bigger they were, the bigger they will be

Substitution, Capture Groups, and ELIZA

Capture Grouping

Every time a capture group is used (i.e., parentheses surround a pattern), the resulting match is stored in a numbered register.

- E.g. /the **(.*)**er they **(.*)**, the **\1**er we **\2**/
- **Example match:** the faster they ran, the faster we ran

Substitution, Capture Groups, and ELIZA

Non-capturing group → (?: pattern)

- When we want to use parentheses for grouping, but don’t want to capture the resulting pattern in a register.
- E.g./(?:some|a few) (people|cats) like some **\1**/
    - Example match
        - **some cats** like some **cats**
    - Example unmatch
        - **some cats** like some **a few**

Substitution, Capture Groups, and ELIZA

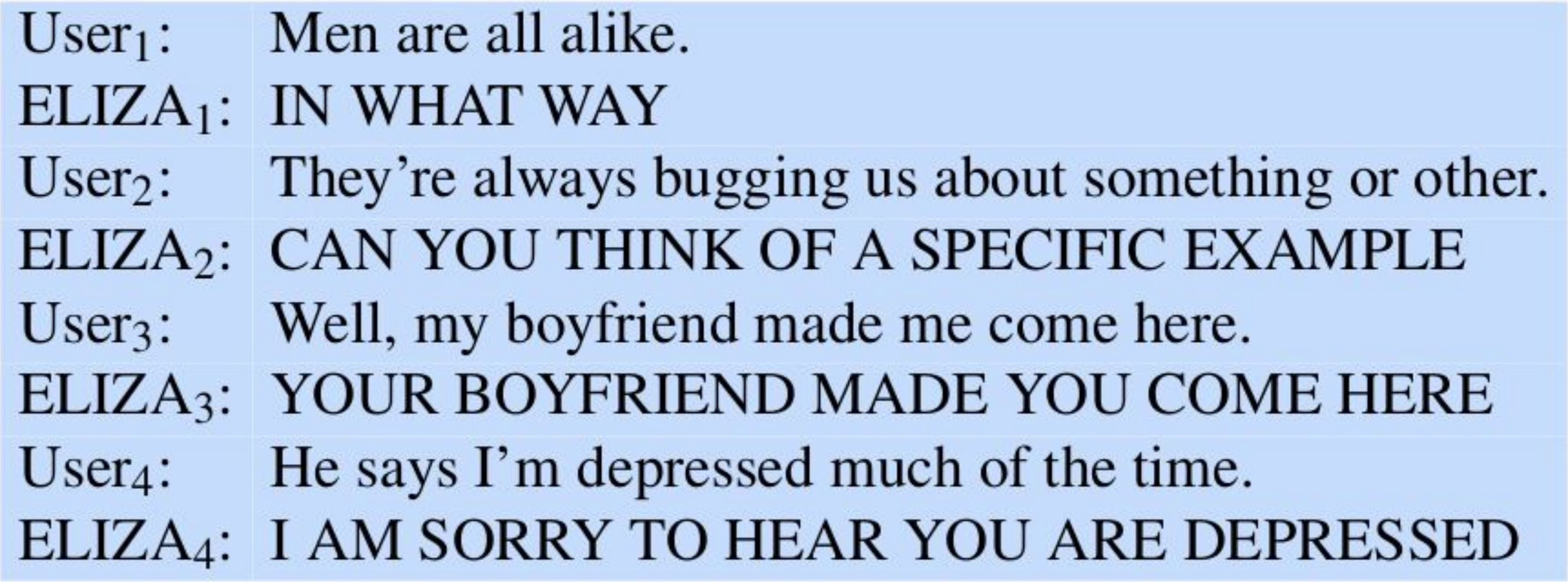
Simple Application: ELIZA

Substitutions and capture groups are very useful in implementing simple chatbots like ELIZA (Weizenbaum, 1966).

Substitution, Capture Groups, and ELIZA

Simple Application: ELIZA

Recall that ELIZA simulates a Rogerian psychologist by carrying on conversations like the following:

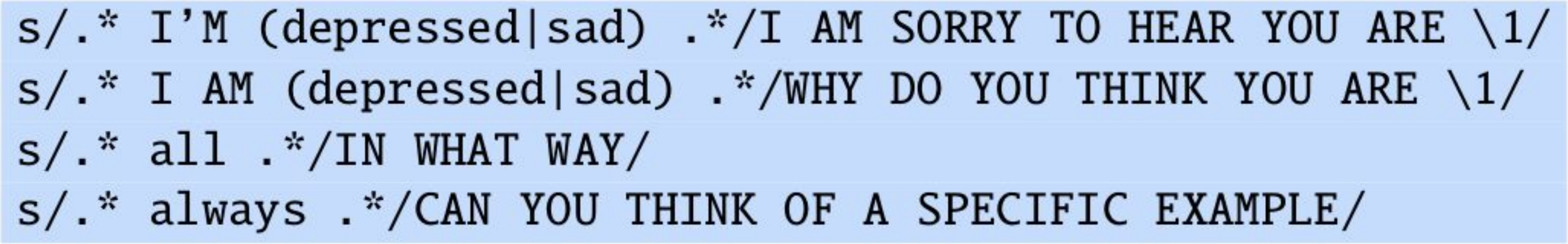


User$_1$:	Men are all alike.
ELIZA$_1$:	IN WHAT WAY
User$_2$:	They're always bugging us about something or other.
ELIZA$_2$:	CAN YOU THINK OF A SPECIFIC EXAMPLE
User$_3$:	Well, my boyfriend made me come here.
ELIZA$_3$:	YOUR BOYFRIEND MADE YOU COME HERE
User$_4$:	He says I'm depressed much of the time.
ELIZA$_4$:	I AM SORRY TO HEAR YOU ARE DEPRESSED

Substitution, Capture Groups, and ELIZA

Simple Application: ELIZA

How ELIZA works



```
s/.* I’M (depressed|sad) .*/I AM SORRY TO HEAR YOU ARE \1/
s/.* I AM (depressed|sad) .*/WHY DO YOU THINK YOU ARE \1/
s/.* all .*/IN WHAT WAY/
s/.* always .*/CAN YOU THINK OF A SPECIFIC EXAMPLE/
```

Research Themes

1. Regular Expressions
2. Universal Programming Languages
3. RSA Public-Key Cryptography

COM1013
INTRODUCTION TO COMPUTER SCIENCE

Lecturer: Begüm MUTLU BİLGE, PhD

begummutlubilge+com1013@gmail.com (recommended)
bmbilge@ankara.edu.tr