Regular Expressions (RE)

- Regular expression (RE): A formula (in a special language) that is used for specifying simple classes of strings.
- String: A sequence of alphanumeric characters (letters, numbers, spaces, tabs, and punctuation).
- Can be used to specify search strings as well as to define a language in a formal way.
- Search requires a pattern to search for, and a corpus of texts to search through.
  - Search through corpus and return all texts that contain pattern.

RE Patterns

- The search string can consist of single character or a sequence of characters.

RE	String matched
/woodchucks/	“interesting links to woodchucks and lemurs”
/a/	“Sarah Ali stopped by Mona’s”
/Alice says,/	“My gift please,” Alice says,”
/book/	“all our pretty books”
/!/	“Leave him behind!” said Sam

RE Disjunctions

- Regular Expressions are case sensitive.
- The string of characters inside [ ] specify 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"

RE Range

- How to conveniently specify *any capital letters ?*
- Use brackets [ ] with the dash (-) to specify any one character in a **range**
- [2-5] – specifies any one of 2, 3, 4, or 5

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"

RE Negation

- Uses of the **caret** ^ for negation or just to mean ^
- ^ symbol is first after open square brace [, the resulting pattern is *negated*

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

RE Cleany star

- Regular expression allows repetition of things.
- Kleene star – zero or more occurrences of previous character or expressions.
- **Kleene *** ----- `/baaa*!/` --- baa!, baaa!, baaaa! .....
- Kleene + – one or more of the previous character
- **Kleene +** ---- `/[0-9]+/` specifies "a sequence of digits"
- Use period **`/./`** to specify any character – a wildcard that matches any single character (except a carriage return)

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

RE Cleany star

RE	Match	Example Patterns Matched
`woodchucks?`	woodchuck or woodchucks	"woodchuck"
`colou?r`	color or colour	"colour"

RE	Description
/a*/	Zero or more a's
/a+/	One or more a's
/a?/	Zero or one a's
/cat|dog/	'cat' or 'dog'
/^cat$/	A line containing only 'cat'
/\bun\B/	Beginnings of longer strings starts by 'un'

RE Anchors, Boundaries

- The caret ^ matches the start of a line.
- The dollar sign $ matches the end of a line.
- Ex: `/^The boat\.$/` matches a line that contains *The boat.*
- \b matches a word boundary while \B matches a non-boundary
- Ex: `/\b55\b/` matches the string: *There are 55 bottles of honey*

  but not *There are 255 bottles of honey*

RE Disjunction, Grouping

- The pipe symbol | is called the disjunction operator
- Example: `/food|wood/` matches either the string food or the string wood
- What is the pattern for matching both the string *puppy* and *puppies*?
- `/puppy|ies/` --> match the strings puppy and ies hence wrong
- The string puppy take precendece over the pipe operator
- Use the parenthesis (and) to make the disjunction (|) apply only to a specific pattern
- `/pupp(y|ies)/` --> match the strings `puppy` and `puppies`

RE Operator Precedence

- Kleene* operator applies by default only to a single character, not a whole sequence.
- Ex: Write a pattern to match the string:
  Column 1 Column 2 Column 3
- `/Column_[0-9]+_*/` matches a column followed by any number of spaces
- The star applies only to the space _ that precedes it, not a whole sequence
- `/(Column_[0-9]+_)*/` --> match the word Column followed by a number, the whole pattern repeated any number of times

RE Operator Precedence

- Parenthesis `( )`
- Counters `* + ? { }`
- Sequences and anchors `the ^my end$`
- Disjunction `|`
- Counters have higher precedence than sequences
  - `/the*/` matches *theeeee* but not *thethe*
- Sequences have a higher precedence than disjunction
  - `/cooky|ies/` matches *cooky* or *ies* but not *cookies*

RE – A Simple Example

- Write a RE to match the English article *the* from the following:
  the
  The
  the124
  @the_
  The – new line

- `/the/`

- missed ‘The’

RE – A Simple Example

- Write a RE to match the English article *the*
- `/the/` missed ‘The’
- `/[tT]he/`
- Need The or the not *the* in ‘others’. Include word boundary

RE – A Simple Example

- Write a RE to match the English article *the*
- `/the/` missed ‘The’
- `/[tT]he/` included *the* in ‘others’
- `/\b[tT]he\b/`
- Perl – word is a sequence of letters, digits and underscores
- Need 'the' from ‘the25’ or ‘the_’

RE – A Simple Example

- Write a RE to match the English article *the*
- `/the/` missed ‘The’
- `/[tT]he/` included *the* in ‘others’
- `/\b[tT]he\b/` missed ‘the25’ ‘the_’
- Make sure no alphabetic letters on either side of *the*
- `/[^a-zA-Z][tT]he[^a-zA-Z]/`
- Issue: won't find the word *The* when it begins the line.

RE – A Simple Example

- Write a RE to match the English article *the*
- `/the/` missed ‘The’
- `/[tT]he/` included *the* in ‘others’
- `/\b[tT]he\b/` missed ‘the25’ ‘the_’
- `/[^a-zA-Z][tT]he[^a-zA-Z]/` missed ‘The’ at the beginning of a line
- Specify that before the *the* we require either the beginning-of-line or non-alphabetic character and the same at end.
- `/(^|[^a-zA-Z])[tT]he([^a-zA-Z]|$)/`

RE – A Complex Example

- Exercise: Write a regular expression that will match
  - “any PC with more than 500MHz and 32 Gb of disk space for less than $1000”
  - First consider RE for prices

RE – A Complex Example

- `/$[0-9]+/` # whole dollars
- What about $155.55 ?
- Deal with fraction of dollars

RE – A Complex Example

- `/$[0-9]+/` # whole dollars
- `/$[0-9]+\.[0-9][0-9]/` # fractions of dollars
- This pattern only allows $155.55 but not $155
- Make cents optional and word boundary

RE – A Complex Example

- `/$[0-9]+/` # whole dollars
- `/$[0-9]+\.[0-9][0-9]/` # fractions of dollars
- `/$[0-9]+(\.[0-9][0-9])?/` # cents optional
- `/\b$[0-9]+(\.[0-9][0-9])?\b/` # word boundary
- Specification for processor speed (in megahertz=MHz or gigahertz=GHz)?

RE – A Complex Example

- `/$[0-9]+/` # whole dollars
- `/$[0-9]+\.[0-9][0-9]/` # fractions of dollars
- `/$[0-9]+(\.[0-9][0-9])?/` # cents optional
- `/\b$[0-9]+(\.[0-9][0-9])?\b/` # word boundary
- Specification for processor speed (in megahertz=MHz or gigahertz=GHz)?
- `/\b[0-9]+_*(MHz|[Mm]egahertz|GHz|[Gg]igahertz)\b/`
- `/_*/` mean “zero or more spaces”
- Memory size?

RE – A Complex Example

- `/$[0-9]+/` # whole dollars
- `/$[0-9]+\.[0-9][0-9]/` # fractions of dollars
- `/$[0-9]+(\.[0-9][0-9])?/` # cents optional
- `/\b$[0-9]+(\.[0-9][0-9])?\b/` # word boundary
- Specification for processor speed (in megahertz=MHz or gigahertz=GHz)?
- `/\b[0-9]+_*(MHz|[Mm]egahertz|GHz|[Gg]igahertz)\b/`
- Memory size: `/\b[0-9]+_*(Mb|[Mm]egabytes?)\b/`
- Allow gigabyte fractions like 5.5Gb

RE – A Complex Example

- `/$[0-9]+/` # whole dollars
- `/$[0-9]+\.[0-9][0-9]/` # fractions of dollars
- `/$[0-9]+(\.[0-9][0-9])?/` # cents optional
- `/\b$[0-9]+(\.[0-9][0-9])?\b/` # word boundary
- Specification for processor speed (in megahertz=MHz or gigahertz=GHz)?
- `/\b[0-9]+_*(MHz|[Mm]egahertz|GHz|[Gg]igahertz)\b/`
- Memory size: `/\b[0-9]+_*(Mb|[Mm]egabytes?)\b/`
- `/\b[0-9](\.[0-9]+)?_*(Gb|[Gg]igabytes?)\b/`
- Operating system and Vendor ?

RE – A Complex Example

- `/$[0-9]+/` # whole dollars
- `/$[0-9]+\.[0-9][0-9]/` # fractions of dollars
- `/$[0-9]+(\.[0-9][0-9])?/` # cents optional
- `/\b$[0-9]+(\.[0-9][0-9])?\b/` # word boundary
- Speed : `/\b[0-9]+_*(MHz|[Mm]egahertz|GHz|[Gg]igahertz)\b/`
- Memory size: `/\b[0-9]+_*(Mb|[Mm]egabytes?)\b/`
- `/\b[0-9](\.[0-9]+)?_*(Gb|[Gg]igabytes?)\b/`
- Vendor: `/\b(Win95|Win98|WinNT|Windows_*(NT|95|98|2000)?)\b/`
- `/\b(Mac|Macintosh|Apple)\b/`

RE – Advanced Operators

- Useful aliases for common ranges

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

Figure 2.6 Aliases for common sets of characters.

RE – Advanced Operators

- RE operators for counting
- /{3}/ means “exactly 3 occurrences of the previous character or expression”

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

RE – Advanced Operators

- Some characters that need to be backlashed

RE	Match	Example Patterns Matched
*	an asterisk "*"	"K*A*P*L*A*N"
\.	a period "."	"Dr. Livingston, I presume"
\?	a question mark	"Would you light my candle?"
\n	a newline	
\t	a tab	

RE – Substitution

- Substitution allows a string characterized by a regular expression to be replaced by another string: `s/regexp1/pattern/`
- Ex: `s/colour/color/`
- Put () around first pattern, and use number operator \1 in second pattern to refer back.
- Ex: `s/([0-9]+)/<\1>/`
- The parenthesis and number operators can also be used to specify that a certain string or expression must occur twice in the text
- `/the (.*)er they were, the \1er they will be/`
- This match with: *The bigger they were, the bigger they will be*

RE – Memory

- The number operator can be used with other numbers:
- `/the (.*)er they (.*), the \1er they \2/`
- This match with: *The bigger they were, the bigger they were*
- These numbered memories are called registers
- Substitutions using memory are useful in natural-language understanding program like ELIZA

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

RE – ELIZA

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

```
s/\bmy\b/YOUR/g
s/\bI('m|am)\b/YOU ARE/g
s/.* YOU ARE (depressed|sad) .*/I AM SORRY TO HEAR YOU ARE \1/
s/.* YOU ARE (depressed|sad) .*/WHY DO YOU THINK YOU ARE \1/
s/.* all .*/IN WHAT WAY/
s/.* always .*/CAN YOU THINK OF A SPECIFIC EXAMPLE/
```

RE – ELIZA style

Step 1: replace first person with second person references

```
s/\bI('m| am)\b /YOU ARE/g
s/\bmy\b /YOUR/g
S/\bmine\b /YOURS/g
```

Step 2: use additional regular expressions to generate replies

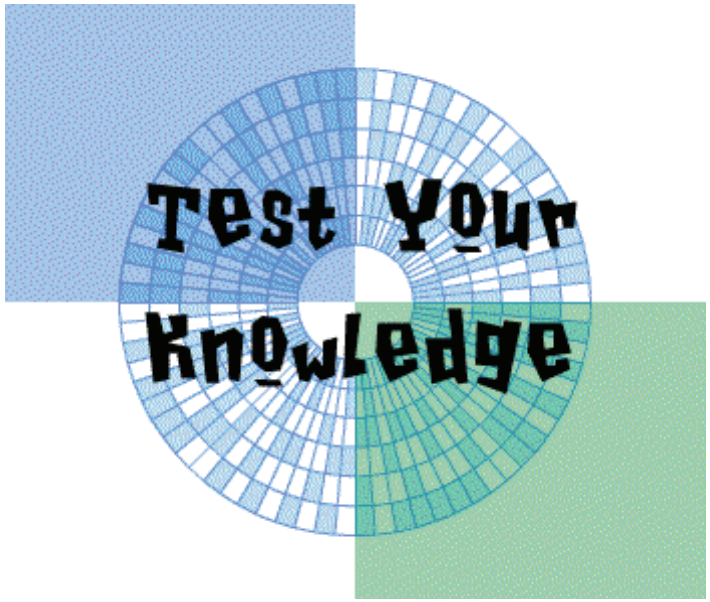
```
s/.* YOU ARE (depressed|sad) .*/I AM SORRY TO HEAR YOU ARE \1/
s/.* YOU ARE (depressed|sad) .*/WHY DO YOU THINK YOU ARE \1/
s/.* all .*/IN WHAT WAY/
s/.* always .*/CAN YOU THINK OF A SPECIFIC EXAMPLE/
```

Step 3: use scores to rank possible transformations

RE – Summary

- Basic regular expression patterns
- Perl-based syntax (slightly different from other notations for regular expressions)
- Disjunctions **[abc]**
- Ranges **[A-Z]**
- Negations **[^Ss]**
- Optional characters **?, +** and *****
- Wild cards **.**
- Anchors **^** and **$**, also **\b** and **\B**
- Disjunction, grouping, and precedence **|, ( )**
- Substitution **s/pattern1/pattern2/**
- Register or Memory **s/pattern1/\1/**

NLP



Write RE for
the set of all lowercase alphabetic strings ending in a *b*

Tips

Write a Perl code to evaluate the RE

Reference

- Speech and Language Processing, Daniel Jurfsky and James H. Martin
- http://perldoc.perl.org/perlretut.html

Thank You!