Lecture 2: Document representation and String processing

COMP90049
Knowledge Technologies

Sarah Erfani and Karin Verspoor, CIS

Semester 2, 2018

THE UNIVERSITY OF MELBOURNE

Unstructured data

- Data without regular, decomposable internal structure
- Examples: blogs, MP3 files, JPEG files
- In practice, most data has *some* structure to it (e.g. track titles in MP3s, document fields in PDF files)

Structured data

- Data which strictly conforms to a schema
- Consistency of data guaranteed by its origins in backend DBs
- Examples: ABN lookup, library catalogues

Semi-Structured data

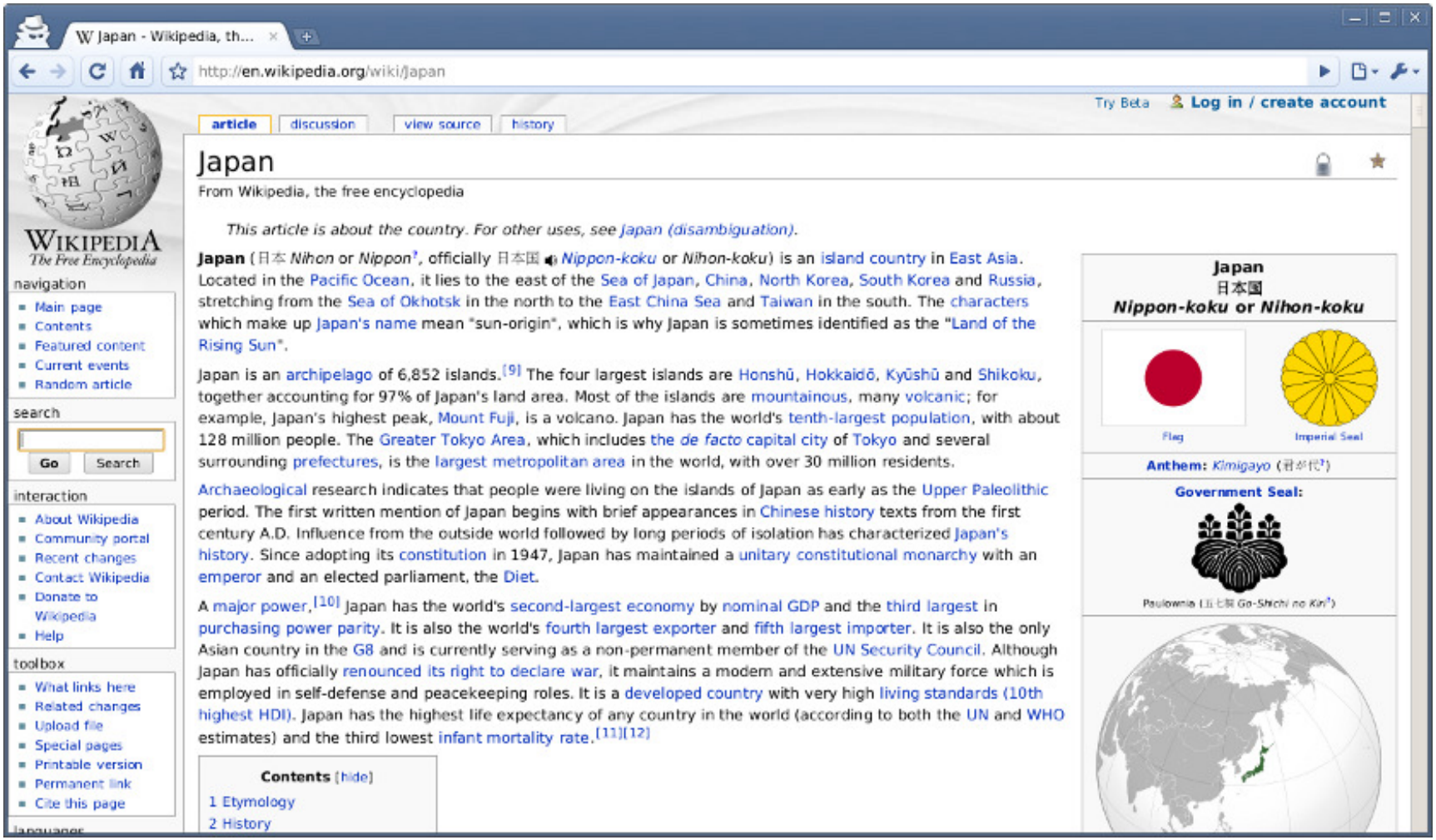
- Data which conforms in part to a schema
  - irregular or incomplete data
  - data which can change in format rapidly and unpredictably
- Examples: Wikipedia entries, BibTeX records

```
@InProceedings{Gulli:Signorini:2005,
  author =    {Antonio Gulli and Alessio Signorini},
  title =    {The Indexable Web is more than 11.5 billion pages},
  booktitle = {Proceedings of the 14th International World Wide
  year =  2005,
  address = {Chiba, Japan}
}
```

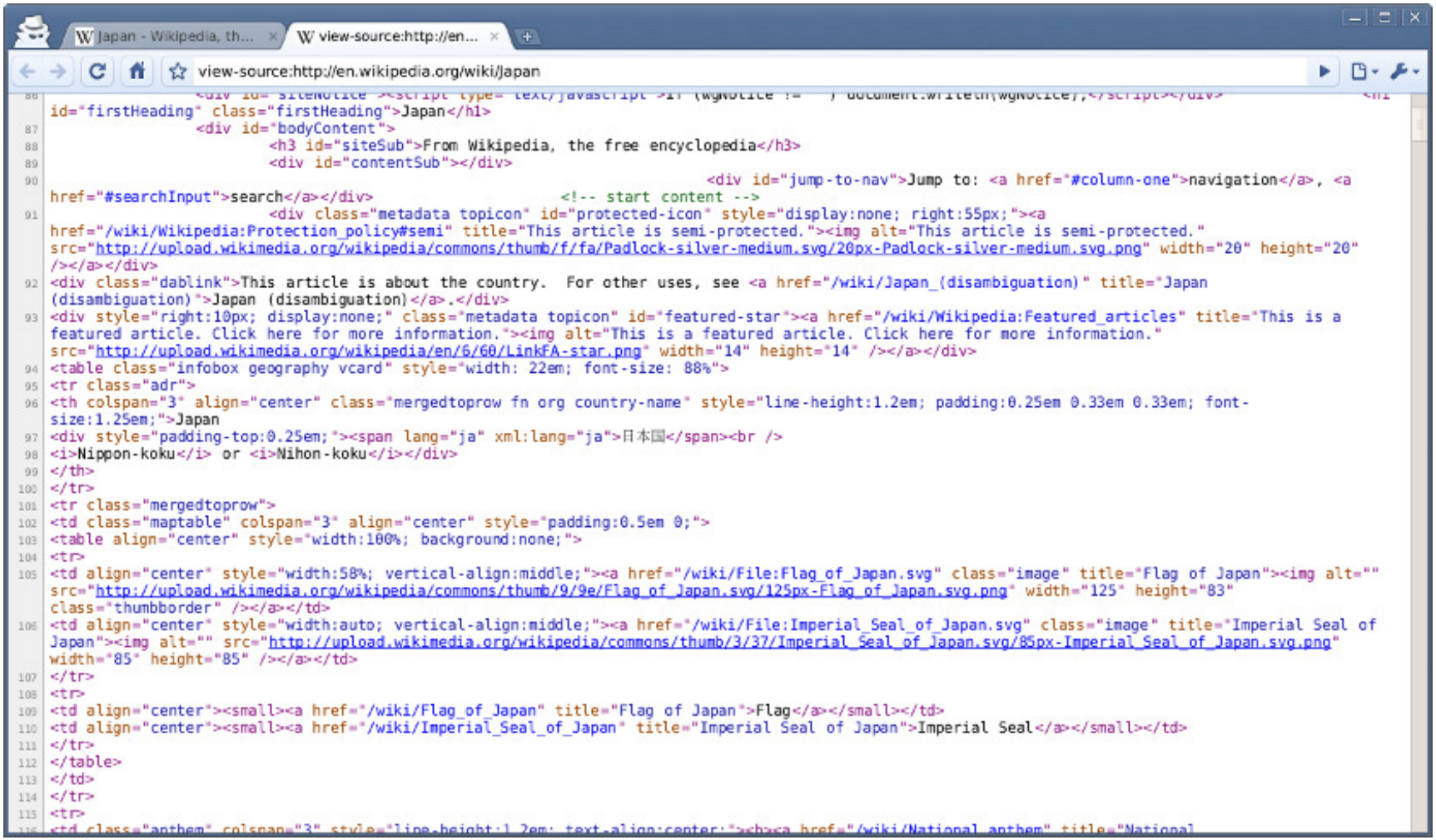
Un- or Semi- or Structured?

- Web pages
- Excel spreadsheet
- Electronic Health Record
- Email
- Video
- Student marks database

Text on the Web: What we see



Text on the Web: What the computer sees



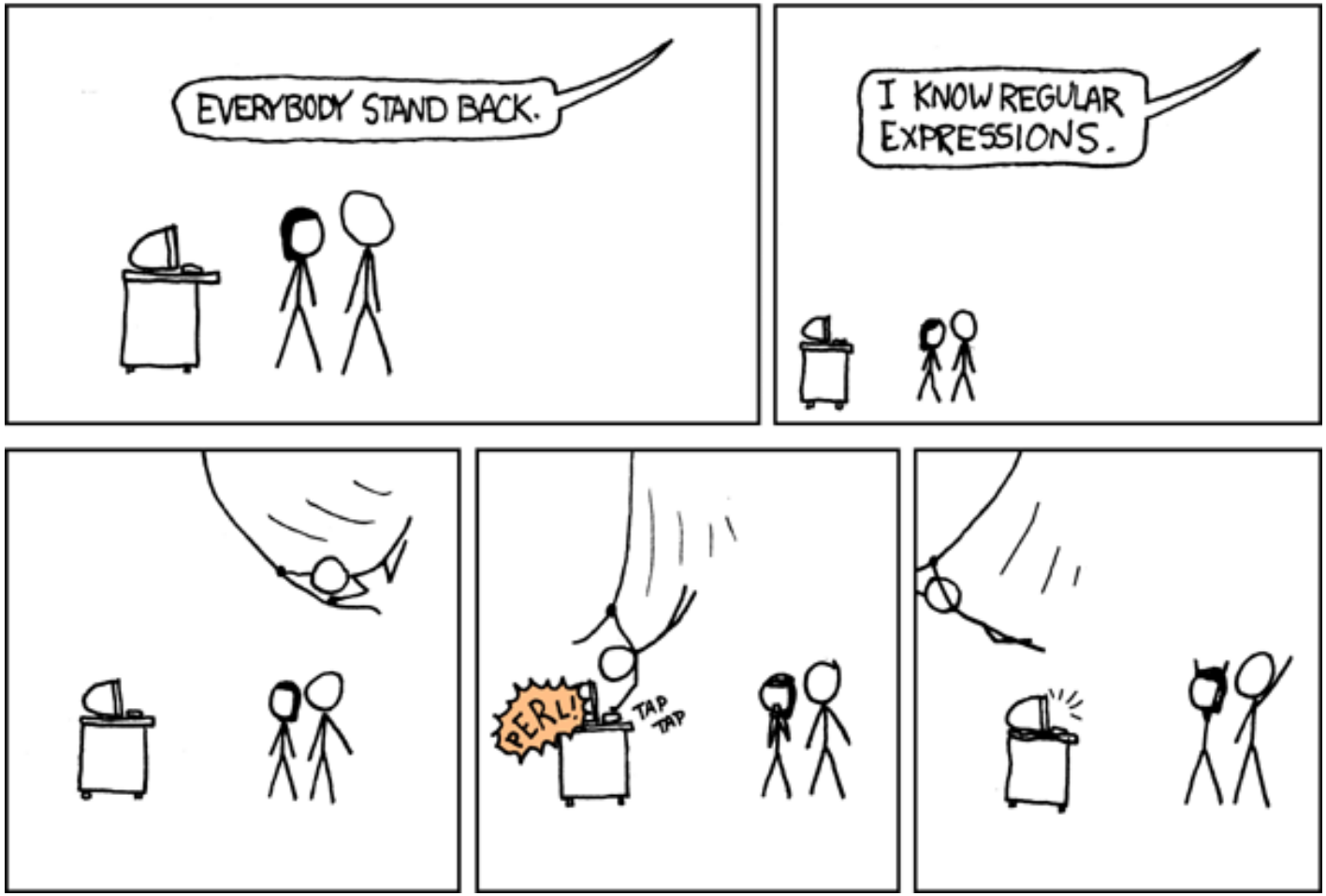

```
id="firstHeading" class="firstHeading">Japan</h1>
87                <div id="bodyContent">
88                        <h3 id="siteSub">From Wikipedia, the free encyclopedia</h3>
89                        <div id="contentSub"></div>
90                                                                <div id="jump-to-nav">Jump to: <a href="#column-one">navigation</a>, <a
href="#searchInput">search</a></div>                    <!-- start content -->
91                        <div class="metadata topicon" id="protected-icon" style="display:none; right:55px;"><a
href="/wiki/Wikipedia:Protection_policy#semi" title="This article is semi-protected."><img alt="This article is semi-protected."
src="http://upload.wikimedia.org/wikipedia/commons/thumb/f/fa/Padlock-silver-medium.svg/20px-Padlock-silver-medium.svg.png" width="20" height="20"
/></a></div>
92 <div class="dablink">This article is about the country.  For other uses, see <a href="/wiki/Japan_(disambiguation)" title="Japan
(disambiguation)">Japan (disambiguation)</a>.</div>
93 <div style="right:10px; display:none;" class="metadata topicon" id="featured-star"><a href="/wiki/Wikipedia:Featured_articles" title="This is a
featured article. Click here for more information."><img alt="This is a featured article. Click here for more information."
src="http://upload.wikimedia.org/wikipedia/en/6/60/LinkFA-star.png" width="14" height="14" /></a></div>
94 <table class="infobox geography vcard" style="width: 22em; font-size: 88%">
95 <tr class="adr">
96 <th colspan="3" align="center" class="mergedtoprow fn org country-name" style="line-height:1.2em; padding:0.25em 0.33em 0.33em; font-
size:1.25em;">Japan
97 <div style="padding-top:0.25em;"><span lang="ja" xml:lang="ja">日本国</span><br />
98 <i>Nippon-koku</i> or <i>Nihon-koku</i></div>
99 </th>
100 </tr>
101 <tr class="mergedtoprow">
102 <td class="maptable" colspan="3" align="center" style="padding:0.5em 0;">
103 <table align="center" style="width:100%; background:none;">
104 <tr>
105 <td align="center" style="width:50%; vertical-align:middle;"><a href="/wiki/File:Flag_of_Japan.svg" class="image" title="Flag of Japan"><img alt=""
src="http://upload.wikimedia.org/wikipedia/commons/thumb/9/9e/Flag_of_Japan.svg/125px-Flag_of_Japan.svg.png" width="125" height="83"
class="thumbborder" /></a></td>
106 <td align="center" style="width:auto; vertical-align:middle;"><a href="/wiki/File:Imperial_Seal_of_Japan.svg" class="image" title="Imperial Seal of
Japan"><img alt="" src="http://upload.wikimedia.org/wikipedia/commons/thumb/3/37/Imperial_Seal_of_Japan.svg/85px-Imperial_Seal_of_Japan.svg.png"
width="85" height="85" /></a></td>
107 </tr>
108 <tr>
109 <td align="center"><small><a href="/wiki/Flag_of_Japan" title="Flag of Japan">Flag</a></small></td>
110 <td align="center"><small><a href="/wiki/Imperial_Seal_of_Japan" title="Imperial Seal of Japan">Imperial Seal</a></small></td>
111 </tr>
112 </table>
113 </td>
114 </tr>
115 <tr>
```

Making sense of data

- Use structure where it is available.
- Use semantics (a schema, meta-data) where it is available.
- Look for bits we 'understand'.

. . . But how?

Regular expressions



From xkcd.com/208, used under Creative Commons Attribution-NonCommercial 2.5 License.

Regular expressions

Regular expressions (regex, regex) are patterns that match character strings.

They can be thought of as describing a set of strings.

- **Search:** Find the strings in a file that contain a substring that matches a given pattern (grep family).

```
> egrep 'rudd' *.txt
> egrep 'col(o|ou)r' *.txt
```

- **Find and replace:** Substitute some new string for the matching substring (sed, vi).

```
s/rudd/gillard/g
s/[dD]og/Canis lupus familiaris/g
```

- **Validate or test:** Check if new string is correct (awk, Python, Perl).

```
$input =~ /gillard/
$input =~ /^[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,4}$/
```

Regular expressions

The four main concepts of regex mirror the four types of structure in imperative programming languages.

Sequence:	`i = 2; j = 3;`	Matching:	`/cat/`
Assignment:	`i = 2;`	Memoization:	(*pattern*)

Selection:	`if A:`
	`  do thing`
	`else:`
	`  do other thing`
		Alternation:	`/cat|dog/`

Repetition:	`while True:` `  i += 1`	Repetition:	`/(cat)*/`

Regular expressions

As the examples above show, regular expressions are a mix of literal characters and command or control characters. For example,

- `a` means "match the character `a`"
- `|` means *or*

`{ } [ ] ( ) ^ $ . | * + ? $ \` are known as *metacharacters* and need to be escaped by a backslash (\) to be used in a literal match; for example,
\$ means "match the character $", and
\\ means "match the character \".

Beware, some tools have different metacharacters. `?` in shells means the same as `.` in standard regex.

And in some cases \ turns a character into a metacharacter.

Here, I sometimes use / as a pattern delimiter. In some tools, it too is a metacharacter.

Matching

The foundation of regex is literal matching:

```
/knowledge/
```

- Each character matches itself.
- Matches are case sensitive.
- Whitespace is significant:
  `/over priced/` won't match "`overpriced`"
- Substrings are uninterpreted; they are not assumed to be whole words or have any specific semantics.
  `/lane/` will match "`planet`"

Another special case is newline. Many tools that incorporate regex are **line-oriented**, and either cannot match across a line break or do so is idiosyncratic ways.

Matching

The wildcard `.` is the most basic metacharacter

- Matches any single character (except a newline); good for crossword puzzles:

```
> egrep '.n.wl.d..' .../local/words.txt
  acknowledge
  acknowledged
  ⋮
```

The anchors `^` and `$` match the start and end of a line or string, respectively.

- `> egrep '^.n.wl.d..$' .../local/words.txt`
  `knowledge`

Alternation

The `|` metacharacter expresses alternation or disjunction

- `/a|b|c/` matches "a", "b", or "c".
- `/cat|dog/` matches "cat" or "dog".
- `/\$(US|AU|CD)/` matches "$US", "$AU", or "$CD".

A note on precedence: the `|` character has low precedence, and the parentheses in the last example are necessary.

Check – what is the difference between:

- `> egrep 'ed|ing$' /usr/share/dict/words`
- `> egrep '(ed|ing)$' /usr/share/dict/words`

Repetition

The precise number of characters to match may be unknown; instead, we specify a repetition construction.

Some repetitions involve an arbitrary number:

- `*`: zero or more of the preceding element
- `?`: zero or one of the preceding element
- `+`: one or more of the preceding element

These are *greedy* – they match as many characters as they can. So `.*` will always match a complete string and `a.*b` will pick up the *last* "`b`" in the string.

Sometimes we care, but only approximately, about number.

- `{n}`: exactly *n* of the preceding element
- `{m,n}`: between *m* and *n* (inclusive) of the preceding element
- `{n,}`: *n* or more of the preceding element
- `{,m}`: up to *m* of the preceding element

For example, `labell?ing` matches "`labeling`", "`labelling`".

Character classes

Sometimes, rather than one particular character or any character, we want to match any of a set of characters.

Some possible character classes:

- `/[Kk]nowledge/`
- `/[aeiou]/` –note that this is equivalent to `/a|e|i|o|u/` or `/(a|e|i|o|u)/`
- `/^\$[0-9]+/`
- `/^[A-Z][a-z]*/`
- `/ [A-Za-z]+ /`

Observe that ranges can be used to denote the character classes.

Observe also that within `[,]`, metacharacters may be used in their literal meaning. For example, in some languages, the class `[\$]` matches "\" or "$".

Negative classes

A second use of the `^` metacharacter is to negate character classes. `/[^A-Za-z]/` matches any non-alpha character.

In some languages, `^` and `-` are the only metacharacters within ranges. (But see the discussion of named classes on the next slide.)

What do these match?

- `/[^0-9]/`
- `/[^"]/`
- `/<[^>]>/`

Named classes

Some character classes are used so frequently that they have names:

- `[0-9]` = `[[:digit:]]` = `\d`
- `[a-zA-Z0-9_]` = `[[:word:]]` = `\w`
- `[\ \t\r\n\f]` = `[[:space:]]` = `\s`

As do their negations:

- `[^0-9]` = `\D`
- `[^a-zA-Z0-9_]` = `\W`
- `[^\ \t\r\n\f]` = `\S`

Beware again: Which named character classes are available and how they are represented depends on the software you use.

Back-references or memoization

Placing a pattern in parentheses leads to the match being stored as a variable.

The first stored pattern has the name `\1`, the *n*th is `\n`. Sadly, there is no way of operating on stored patterns, but they can be accessed for subsequent matching.

Example: What does `/([a-zA-Z]+) +\1/` match?

They are particularly powerful in string substitution.

Example: `s/([A-Z])[a-z]+ ([A-Z][a-z]+)/\1. \2/`

Putting it all together

Now we can parse the regex from earlier on:

```
/^[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,4}$
```

- `^[A-Z0-9._%+-]+`: match one or more of these characters
- `@`: followed by an "@"
- `[A-Z0-9.-]+`: followed by one or more of these characters
- `\.`: followed by a dot
- `[A-Z]{2,4}$`: followed by 2–4 upper case letters, and then end of line

- What do you think this pattern is for?
- How might this pattern be improved?

Programming with patterns

There are several pattern-based programming languages, in particular `Python` and `Perl`. There are also good command-line tools, in particular `sed` and `awk`. (Perl is also used in this way.)

A quick look at `awk` . . .

- Line-oriented; each block of code describes a series of operations to be applied to a line of input. Every line is processed in turn.
- Code is C-like (i.e., Java-like, C++-like).
- Lines of input are parsed into fields, and assigned to variables `$1`, `$2`, `$3`, . . .
- A line of input is only processed if it matches a pattern.
- Fields may be tested to see if they match a pattern.

Programming with patterns

```
Baughman Edward D. <Edward.Baughman@ENRON.com>
Baughman Edward <Edward.Baughman@ENRON.com>
Becker Lorraine <Lorraine.Becker@ENRON.com>
"Beck, Sally" <Sally.Beck@ENRON.com>,
Beck Sally <Sally.Beck@ENRON.com>
bejules@hotmail.com
Ben <Ben.Brasseaux@ENRON.com>
```

This is a complete `awk` program for processing the input above.

```
/<[^ ]*@ENRON[^ ]*>/{
    for( i=1 ; i<=NF ; i++ )
        if( $i ~ /^[A-Za-z]*$/ ) print $i;
}
```

`NF` is a special variable containing the number of fields in the current line. Other variables (e.g., `i`) are created automatically when they are referenced.

Summary

- What are regular expressions and what are they used for?
- What are the main concepts used in regular expressions?
- What kinds of search tasks can and cannot be addressed with regular expressions?

- Consolidate your understanding of the regular expression metacharacters; some useful references:

`docs.python.org/dev/howto/regex.html`

`perldoc perlretut` on any CIS server (or even a Mac!)

`perldoc.perl.org/perlretut.html`

`java.sun.com/docs/books/tutorial/essential/regex/`

Next Lecture: Similarity