LING 473: Day 2

START THE RECORDING

Counting, Unix, Regexes

Combining Counts

- Fundamental principle:

> The cross product of the sets
>
> $\{\, m_0 \,\ldots\, m_{c_m} \}$ and $\{\, n_0 \,\ldots\, n_{c_n} \}$
>
> has $c_m \times c_n$ members

- There are $m \times n$ ways to combine one element from the first set and one element from the second set.

Combining Counts

- If we choose *m* times from the same set of *n* elements (without depletion, "reusing" elements), there are n^m possible combinations.*

- How many four-letter strings can be formed with the English alphabet?

$$26^4 = 456{,}976$$

**This includes all orderings, even between two elements that are the same. E.g., two 'a's can be ordered 2 ways: aa, aa.*

Counting: Combinatorics

- The basics of counting and arranging sequences:

Permutations:
all possible orderings of *n* elements

Combinations:
all (unordered) sets of *k* elements that can be selected from *n* elements

Variations:
all orderings of *k* elements that can be selected from *n* elements.

Permutations

{ o, r, a, n, g, e }

6 choices	a						{ o, r, n, g, e }
5 choices	a	e					{ o, r, n, g }
4 choices	a	e	g				{ o, r, n }
3 choices	a	e	g	n			{ o, r }
2 choices	a	e	g	n	o		{ r }
1 choice	a	e	g	n	o	r	{ }

$6 \times 5 \times 4 \times 3 \times 2 \times 1 =$
$6! = 720$

This is the factorial function, denoted by $n!$

Permutations

- Permutation only rearranges elements. It doesn't do repetition.
- But if the input has repetitions, how do we ignore that when finding the permutation?
- Do the permutation (n!) and then divide by the number of permutations that are identical due to repeated elements.
  - e.g., input: {a, a, b, c}, how many orders?
  - we don't want to count (a, b, c, a) twice by choosing the a's in different orders!

Permutations

- How many items combinations are identical?
  - There's some number of subsets of identical items:

    $\{ a_0, a_1, \ldots, \mathrm{a}_x \}$

    $\{ b_0, b_1, \ldots, b_y \}$

    $\{ c_0, c_1, \ldots, c_z \}$

    or to speak generally:

    $\{ r_{i,0}, r_{i,1}, \ldots, r_{i,m} \}$
  - So we divide by the product of the number of orderings for the sets of identical elements:

$$\prod_i m_i!$$

Permutations

An example:

mississippi

11 letters

repeated group {i, i, i, i} : cardinality 4

repeated group {s, s, s, s} : cardinality 4

repeated group {p, p} : cardinality 2

$$\frac{11!}{4! \times 4! \times 2!} = 34{,}650$$

Combinations

- How many unordered sets of size *k* can be formed from a set of cardinality *n*?
- If we can use elements as many times as we want, we already know how to do this: n^k
- What about with depleting?

Combinations

Let's say we have seven books, and we need to choose just 3 (maybe for vacation). How many different books can we take?

1st choice: 7 books to choose from

2nd choice: 6 books to choose from

3rd choice: 5 books to choose from

$7 \times 6 \times 5 = 210$ book combinations

BUT… ordering doesn't matter. Each set comes in 3! different orderings, so we divide by 3! (=6), thus counting each ordering once.

$210 \div 6 = 35$

Combinations

- Our numerator was $n(n-1)\ldots(n-k+1)$
- The denominator was $k(k-1)\ldots(2)(1)$ or $k!$
- We can rewrite the numerator using factorials:

$$\frac{n!}{(n-k)!} = \frac{n(n-1)\ldots(n-k+1)\cancel{(n-k)}\ldots\cancel{(2)}\cancel{(1)}}{\cancel{(n-k)}\ldots\cancel{(2)}\cancel{(1)}}$$

$$= n(n-1)\ldots(n-k+1)$$

- So the full rewrite looks like:

$$\frac{n!}{(n-k)!\,k!}$$

- This is called the binomial coefficient or the choose function

Combinations

upper index

lower index

$$\binom{n}{k} = \frac{n!}{(n-k)!\,k!}$$

- Binomial coefficient
- Pronounced "*n* choose *k*"
- How many subsets of size *k* can be formed from a set of size *n*

Variations

- Variations are permutations of combinations: ordered sub-collections of size *k* chosen from a set *n*.
- These are just cases where we don't need to cancel out repeated outputs. That is, order *does* matter.
- Choosing elements from the set more than once:

$$n^k$$

- Choosing elements from the set only once:

$$\frac{n!}{(n-k)!}$$

Repetition within a Collection

Ordered or unordered groups?

Distinct elements or repeated elements?

{ set }	unordered, distinct (no repetition)
{ multiset, bag }	unordered, repetition permitted
(tuple)	ordered, repetition permitted

- For combinations (not permutations) we can allow repetition in the *output*
  - Do we want the combinations of “{ a b c } choose 2” to include { a a }—in which case there are 6 output sets—or not—in which case there are only 3: { a b }, { a, c }, { b, c}

Repetition within a Collection

- If we are creating a set from an input, and the input has repeated values, we have to decide how to count repition in the output.
- In *permutations*, we count repetitions separately. Permutations of {a, a, b} = 3!, including two instances of (a a b), (a b a), (b a a).
  - Often we don't want both. Divide by the repetitions.
- In *combinations*, we count repetitions if they are present in the input. "{a, a, b} choose 2" includes two instances of {a, b}.
  - Do we want to count both? Depends: {a, b} is twice as likely as {a, a}: that might be useful.

Permutations v. Variations

		output type	
		permutations	variations with repetition where lower index = input cardinality
input	distinct { a, b, c}	(a, b, c), (a, c, b), (b, a, c), … (c, b, a) $n!$	(a, a, a) (a, a, b) (a, a, c) … (c, c, c) n^n
	has repetition (a, a, b)	(a a b), (a b a) ~~(a a b)~~, (b a a) ~~(a b a)~~, ~~(b a a)~~ $\frac{n!}{\prod_i (m_i!)}$	(a, a, a) (a, a, b) (a, b, a) … (b, b, b) $(\#\text{distinct elements})^n$

Duplicate output tuples

- Input (has repetition)

  (a, a, b)

  Variations (lower index=3) allowing repetition in the output: 27 of them

(a a a) (a a a) (a a b) (a a a) (a a a) (a a b) (a b a)

(a b a) (a b b) (a a a) (a a a) (a a b) (a a a) (a a a)

(a a b) (a b a) (a b a) (a b b) (b a a) (b a a) (b a b)

(b a a) (b a a) (b a b) (b b a) (b b a) (b b b)

There are only 8 different output tuples

Combinatorics Summary

{ a b c }

- Permutation: how many different orderings?

  (a b c) (a c b) (b a c) (b c a) (c a b) (c b a) $n!$

- Combination: how many different subsets (i.e. of 2)?

  { a b } { a c } { b c } $\binom{n}{k}$

  allowing repetition in the output

  { a a } { a b } { a c } { b b } { b c } { c c }

- Variations: how many different ordered subsets (i.e. of 2)?

  (a b) (a c) (b a) (b c) (c a) (c b) $\frac{n!}{(n-k)!}$

  allowing repetition in the output

  (a a) (a b) (a c) (b a) (b b) (b c) (c a) (c b) (c c) n^k

Assignment 1

- Due July 27, 4:40 p.m. (this Thursday, before class!)
- Relatively brief
- Goes over counting, probability
- Turn in via CollectIt

Unix

- Bell Labs, 1969: Thompson, Ritchie, et al.
- Simple model: a UI-less ‘kernel’ provides process, device, and memory management
- Command line shells provide interactive interaction, if required:
  - sh, csh, ksh, bash
- Historical progression of implementations
  - System V, BSD, POSIX, Linux, Mac OS X
- X-Windows: a graphical interface to the kernel
- KDE, Gnome: graphical desktops

Shell Commands: Files

$ ls	list files in the current directory
$ cat	show the contents of a file
$ cp	copy a file
$ mv	move or rename a file
$ rm	delete a file

These are just programs, really.

Since filenames are case sensitive, so are these command names.

Shell Commands: File permissions

$ chmod +x myfile

makes myfile executable (to everyone)

$chmod u+x myfile

makes myfile executable (to the user = owner)

$ chmod 774 myfile

binary: 111111100

set r w x r w x r - -

for owner (u), group (g), and others (o)

Shell Commands: Directories

$ cd	change working directory
$ mkdir	make a new directory
$ rmdir	remove a directory
/	separates directory paths
.	refers to the current directory
..	refers to the parent directory

Unix console control

ctrl-D	end the program, end of stream this is ctrl-Z on DOS
ctrl-S	XOFF, pause the stream (if supported) beware appearance of hang
ctrl-Q	XON, resume the stream (if supported)
ctrl-C	attempt to interrupt the program
ctrl-L	FF, form feed, clear the screen
ctrl-I	HT, horizontal tab
ctrl-M	CR, carriage return (see next slide)
ctrl-J	LF, line feed (see next slide)

Text file line endings

- Different systems use different conventions for line endings in text files (i.e. corpora)
- Since files are migrated between systems, we will always need to handle these differences correctly

System	Line ending convention	ASCII	Unicode	C
Unix (and OS X)	LF	0A	000A	\n
DOS/Windows	CR LF	0D 0A	000D 000A	\r\n
Macintosh (Pre-OS X)	CR	0D	000D	\r

Text Editors

- vi, vim
- emacs
- nano
- pico
- other solutions: local editor with ssh script

Shell scripts

myprog.sh

‘shebang’

```
#!/bin/sh

# the hash mark indicates a comment line
ls /
```

to run this program:

```
$ ./myprog.sh
```

notice the mention of the current directory

Streams and I/O

- Unix uses the concept of ‘streams’ – a series of characters with no (predetermined) end.

0	Standard Input	stdin
1	Standard Output	stdout
2	Standard Error	stderr

Pipes and Redirection

>	redirect output to a new file (overwrites if exists)
1>	redirect output from stdout to a new file (same as >)
2>	redirect output from stderr to a new file
<	redirect input from a file
>>	append output to a new or existing file
&>	redirect stdout and stderr to a new file
|	pipe stdout to the next program as stdin

```
$ls -l | sort
```

Pipes and Redirection

$ more	show output one screen at a time
$ tail	show the last 10 lines of a file
$ cat foo >bar	redirect contents of ‘foo’ to stdout
$ cat foo 1>bar	same thing

$ ./myprog <foo >bar 2>errout

execute ‘myprog,’ pass the contents of ‘foo’ in as standard input, capture standard output to ‘bar,’ and capture standard error to ‘errout’

Text Processing

wc — word count

arguments `-l, -w, -c` count lines, words, characters

sort — general purpose ASCII or ordinal sort

It's not encoding-aware which makes it limited for serious linguistic use

tr — translate (substitute character ranges)

grep — search for matching patterns

sed — stream editor

uniq — remove duplicate lines (from sorted files)

diff — compare text files

Working with data

- Basic definitions:

bit: a single memory cell that can have the value zero or one

this is the fundamental representation of information

byte: a fixed group of 8 (eight) ordered, distinct bits

therefore, a byte has a value between 0 and $(2^8 - 1 = 255)$

MSB: the most-significant-bit in a byte
LSB: the least-significant-bit in a byte

KB: one kilobyte: $2^{10} = 1024$ bytes
MB: one megabyte: $2^{20} = 1{,}048{,}576$ bytes
GB: one gigabyte: $2^{30} = 1{,}073{,}741{,}824$ bytes
4 GB: four gigabytes: $2^{32} = 4{,}294{,}967{,}296$ bytes

8-bit character encodings

- 8-bit: $2^8 = 256$ possible characters

  characters 0-127: usually ASCII, fairly standardized

  characters 128-255: a free-for-all

  - Hundreds of different systems for assigning various characters to the 256 available positions
  - Each of these is a character encoding

A (partial) list of some 8-bit character encodings

IBM037	IBM EBCDIC (US-Canada)
IBM437	OEM United States
IBM500	IBM EBCDIC (International)
ASMO-708	Arabic (ASMO 708)
DOS-720	Arabic (DOS)
ibm737	Greek (DOS)
ibm775	Baltic (DOS)
ibm850	Western European (DOS)
ibm852	Central European (DOS)
IBM855	OEM Cyrillic
ibm857	Turkish (DOS)
IBM00858	OEM Multilingual Latin I
IBM860	Portuguese (DOS)
ibm861	Icelandic (DOS)
DOS-862	Hebrew (DOS)
IBM863	French Canadian (DOS)
IBM864	Arabic (864)
IBM865	Nordic (DOS)
cp866	Cyrillic (DOS)
ibm869	Greek, Modern (DOS)
IBM870	IBM EBCDIC (Multilingual Latin-2)
windows-874	Thai (Windows)
cp875	IBM EBCDIC (Greek Modern)
shift_jis	Japanese (Shift-JIS)
gb2312	Chinese Simplified (GB2312)
ks_c_5601-1987	Korean
big5	Chinese Traditional (Big5)
IBM1026	IBM EBCDIC (Turkish Latin-5)
IBM01047	IBM Latin-1
IBM01140	IBM EBCDIC (US-Canada-Euro)
IBM01141	IBM EBCDIC (Germany-Euro)
IBM01142	IBM EBCDIC (Denmark-Norway-Euro)
IBM01143	IBM EBCDIC (Finland-Sweden-Euro)
IBM01144	IBM EBCDIC (Italy-Euro)
IBM01145	IBM EBCDIC (Spain-Euro)
IBM01146	IBM EBCDIC (UK-Euro)
IBM01147	IBM EBCDIC (France-Euro)
IBM01148	IBM EBCDIC (International-Euro)
IBM01149	IBM EBCDIC (Icelandic-Euro)
windows-1250	Central European (Windows)
windows-1251	Cyrillic (Windows)
Windows-1252	Western European (Windows)
windows-1253	Greek (Windows)
windows-1254	Turkish (Windows)
windows-1255	Hebrew (Windows)
windows-1256	Arabic (Windows)
windows-1257	Baltic (Windows)
windows-1258	Vietnamese (Windows)
Johab	Korean (Johab)
macintosh	Western European (Mac)
x-mac-japanese	Japanese (Mac)
x-mac-chinesetrad	Chinese Traditional (Mac)
x-mac-korean	Korean (Mac)
x-mac-arabic	Arabic (Mac)
x-mac-hebrew	Hebrew (Mac)
x-mac-greek	Greek (Mac)
x-mac-cyrillic	Cyrillic (Mac)
x-mac-chinesesimp	Chinese Simplified (Mac)
x-mac-romanian	Romanian (Mac)
x-mac-ukrainian	Ukrainian (Mac)
x-mac-thai	Thai (Mac)
x-mac-ce	Central European (Mac)
x-mac-icelandic	Icelandic (Mac)
x-mac-turkish	Turkish (Mac)
x-mac-croatian	Croatian (Mac)
x-Chinese-CNS	Chinese Traditional (CNS)
x-cp20001	TCA Taiwan
x-Chinese-Eten	Chinese Traditional (Eten)
x-cp20003	IBM5550 Taiwan
x-cp20004	TeleText Taiwan
x-cp20005	Wang Taiwan
x-IA5	Western European (IA5)
x-IA5-German	German (IA5)
x-IA5-Swedish	Swedish (IA5)
x-IA5-Norwegian	Norwegian (IA5)
us-ascii	US-ASCII
x-cp20261	T.61
x-cp20269	ISO-6937
IBM273	IBM EBCDIC (Germany)
IBM277	IBM EBCDIC (Denmark-Norway)
IBM278	IBM EBCDIC (Finland-Sweden)
IBM280	IBM EBCDIC (Italy)
IBM284	IBM EBCDIC (Spain)
IBM285	IBM EBCDIC (UK)
IBM290	IBM EBCDIC (Japanese katakana)
IBM297	IBM EBCDIC (France)
IBM420	IBM EBCDIC (Arabic)
IBM423	IBM EBCDIC (Greek)
IBM424	IBM EBCDIC (Hebrew)
x-EBCDIC-KoreanExtended	IBM EBCDIC (Korean Extended)
IBM-Thai	IBM EBCDIC (Thai)
koi8-r	Cyrillic (KOI8-R)
IBM871	IBM EBCDIC (Icelandic)
IBM880	IBM EBCDIC (Cyrillic Russian)
IBM905	IBM EBCDIC (Turkish)
IBM00924	IBM Latin-1
EUC-JP	Japanese (JIS 0208-1990 and 0212-1990)
x-cp20936	Chinese Simplified (GB2312-80)
x-cp20949	Korean Wansung
cp1025	IBM EBCDIC (Cyrillic Serbian-Bulgarian)
koi8-u	Cyrillic (KOI8-U)
iso-8859-1	Western European (ISO)
iso-8859-2	Central European (ISO)
iso-8859-3	Latin 3 (ISO)
iso-8859-4	Baltic (ISO)
iso-8859-5	Cyrillic (ISO)
iso-8859-6	Arabic (ISO)
iso-8859-7	Greek (ISO)
iso-8859-8	Hebrew (ISO-Visual)
iso-8859-9	Turkish (ISO)
iso-8859-13	Estonian (ISO)
iso-8859-15	Latin 9 (ISO)
x-Europa	Europa
iso-8859-8-i	Hebrew (ISO-Logical)
iso-2022-jp	Japanese (JIS)
csISO2022JP	Japanese (JIS-Allow 1 byte Kana)
iso-2022-jp	Japanese (JIS-Allow 1 byte Kana - SO/SI)
iso-2022-kr	Korean (ISO)
x-cp50227	Chinese Simplified (ISO-2022)
euc-jp	Japanese (EUC)
EUC-CN	Chinese Simplified (EUC)
euc-kr	Korean (EUC)
hz-gb-2312	Chinese Simplified (HZ)
GB18030	Chinese Simplified (GB18030)
x-iscii-de	ISCII Devanagari
x-iscii-be	ISCII Bengali
x-iscii-ta	ISCII Tamil
x-iscii-te	ISCII Telugu
x-iscii-as	ISCII Assamese
x-iscii-or	ISCII Oriya
x-iscii-ka	ISCII Kannada
x-iscii-ma	ISCII Malayalam
x-iscii-gu	ISCII Gujarati
x-iscii-pa	ISCII Punjabi

8-bit encodings

- Great for 1968
  - why?
- Not so great for 2015
  - why?

Unicode

- Unicode uses 16-bits to store each character
  $2^{16} = 65535$ possible characters
  - Major languages are well represented
  - Standard assignments: no conflicting characters
  - Combining characters for accents, ligatures
  - Unicode layout engines are more intelligent than just a monospace console
  - Compatible: characters 0-127 are the same as ASCII
- A single Unicode character is called a code point
- Unicode text is a stream of code points

UTF-8

- Unicode is nice, but hey, my documents are 99% English. Why do they have to be twice as big?
- 8-bit Unicode Transformation Format (UTF-8) uses a variable number of bytes to encode Unicode characters
- In fact, if you only use ASCII, the UTF-8 stream looks like an 8-bit ASCII stream
- 1, 2, 3, or 4 bytes per character are used
  - this means that some Unicode streams get *larger* using UTF-8
  - This will probably be true for alphabets/languages other than:

    Extended Latin alphabet, Romance languages, Greek, Cyrillic, Coptic, Armenian, Hebrew, Arabic, Syriac, Tāna

How does UTF-8 work?

Unicode range		Encoded bytes	Example
Hex	Binary		
U+0000 to U+007F	00000000 to 01111111	0xxxxxxx	'$' U+0024 = 00100100 → 00100100 → 0x24
U+0080 to U+07FF	00000000 10000000 to 00000111 11111111	110yyyxx 10xxxxxx	'¢' U+00A2 = 00000000 10100010 → 11000010 10100010 → 0xC2 0xA2
U+0800 to U+FFFF	00001000 00000000 to 11111111 11111111	1110yyyy 10yyyyxx 10xxxxxx	'€' U+20AC = 00100000 10101100 → 11100010 10000010 10101100 → 0xE2 0x82 0xAC
U+010000 to U+10FFFF	00000001 00000000 00000000 to 00010000 11111111 11111111	11110zzz 10zzyyyy 10yyyyxx 10xxxxxx	'𤭢' U+024B62 = 00000010 01001011 01100010 → 11110000 10100100 10101101 10100010 → 0xF0 0xA4 0xAD 0xA2

Using HTML <meta> tag to specify encoding

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
```

This is nice, but you still have to:

- Save the file using the specified encoding
  - This requires using an editor that is capable of doing so
- Configure the web server to send a matching HTTP header
  - This is an out-of-band mechanism for specifying the content encoding of every single web page

```
HTTP/1.1 200 OK
Date: Mon, 23 May 2005 22:38:34 GMT
Server: Apache/1.3.3.7 (Unix)  (Red-Hat/Linux)
Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT
Etag: "3f80f-1b6-3e1cb03b"
Accept-Ranges: bytes
Content-Length: 438
Connection: close
Content-Type: text/html; charset=UTF-8
```

Endianness

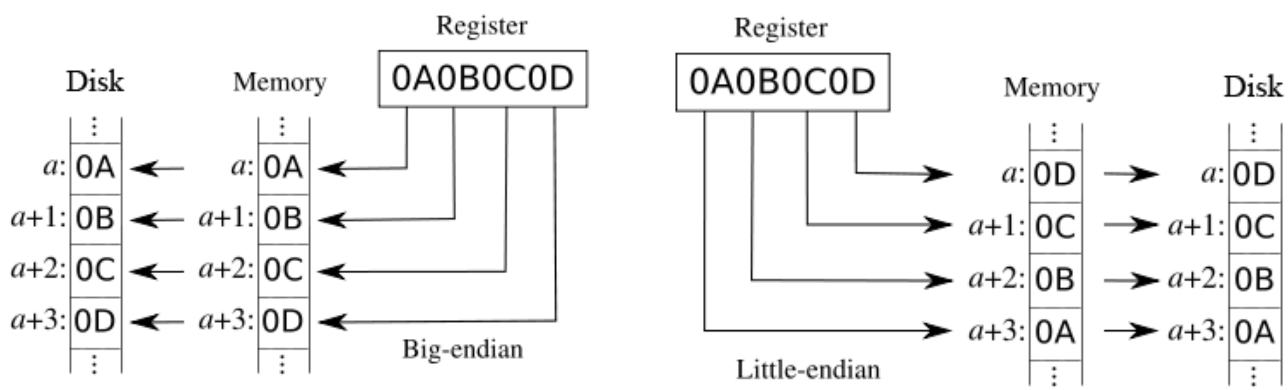
- The layout of 16-bit or 32-bit values in memory is microprocessor dependent
  - nowadays, this is not an issue for bytes
- If a disk file is just a copy of a memory image, then this difference can persist in the file
  - big endian:

    most-significant-byte … least-significant-byte
  - little endian:

    least-significant-byte … most-significant-byte

Why this matters to computational linguistics

- Since unicode uses 16-bits per character, endianness sometimes matters



- UTF-8 is defined a stream of bytes, however, so it is not affected by this issue

Byte-order Mark (BOM)

- Solving the 'endian-ness' problem for Unicode files
- But also used as in-band means for distinguishing Unicode file formats UTF-8 and UTF-16
- Infrequently used, but important for computational linguists to be aware of
- Examine the first few bytes of a text file for the BOM

Encoding	Representation (hexadecimal)	Representation (decimal)	Representation (ISO-8859-1)
UTF-8	EF BB BF	239 187 191	ï»¿
UTF-16 (BE)	FE FF	254 255	þÿ
UTF-16 (LE)	FF FE	255 254	ÿþ

Firefox browser does not like to see a BOM at the top of an HTML file

Programming language support for encodings

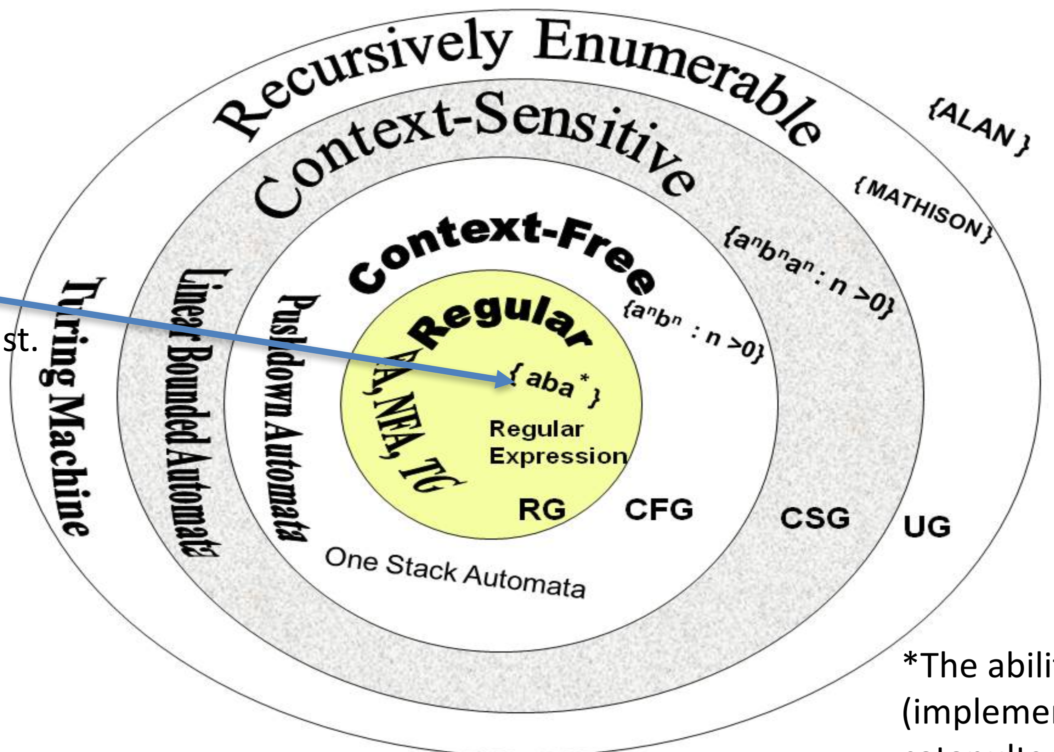
```
String s = "สวัสดีครับ";       // C# strings are always unicode
int i = s.Length;             // number of code points: 10
Char ch = s[0];               // always a 16-bit character.
                              // In this case the value is 3626 (U+0E2A)

Byte[] bytes = Encoding.GetEncoding("TIS-620").GetBytes(s);
i = bytes.Length;             // number of bytes: 10
byte b = bytes[0];            // a byte. In this case, the value is 202 (\xCA)

bytes = Encoding.UTF8.GetBytes(s);
i = bytes.Length;             // number of bytes: 30

s = Encoding.UTF8.GetString(bytes);     // back to the original string
```

Syntax Machines

- Grammars (in the generic sense) are machines for manipulating strings.
- Less powerful grammars cannot distinguish between as many strings, but are faster
- More powerful grammars can distinguish between more strings (rule things out), and are slower
- The study of grammars and computational complexity is part of automata theory

Syntax Machines

Regular expressions
are here.*
Not powerful, but fast.



*The ability to remember strings
(implemented in most regex languages)
catapults regexes beyond regular language

Regular Expressions

- A syntax for matching patterns in text.
  - Big theoretical contributors: Stephen Kleene, Ken Thompson
  - Fast, but simple (and limited)

Basic RegEx

^	matches the start of a line
$	matches the end of a line
.	matches any one character (except newline)
[*xyz*]	matches any one character from the set
[^*pdq*]	matches any one character not in the set
|	or (accepts either its left or its right side)
\	escape to specify special characters (e.g., ^ $ \ [ ] ( ))

anything else: must match exactly

More RegEx

*	accepts zero or more of the preceding element (Kleene star)
	this is the canonical ‘greedy’ operator
?	accepts zero or one of the preceding element(s)
+	accepts one or more of the preceding element(s)
{*n*}	accepts *n* of the preceding element(s)
{*n*,}	accepts *n* or more of the preceding element(s)
{*n*,*m*}	accepts *n* to *m* of the preceding element(s)

(*pattern*) defines a capture group which can be referred to later via \1
(if there are two ()s, \1 & \2, etc)

RegEx Examples

- Find the English stops followed by a liquid

  grep [PTKBDGptkbdg][lr]

- Find any two vowels together

  grep [aeiou][aeiou]

- Find the same letter, repeated

  egrep '([a-z])\1'

- Lines where sentences end with ‘to’

  egrep '(|^)to.'

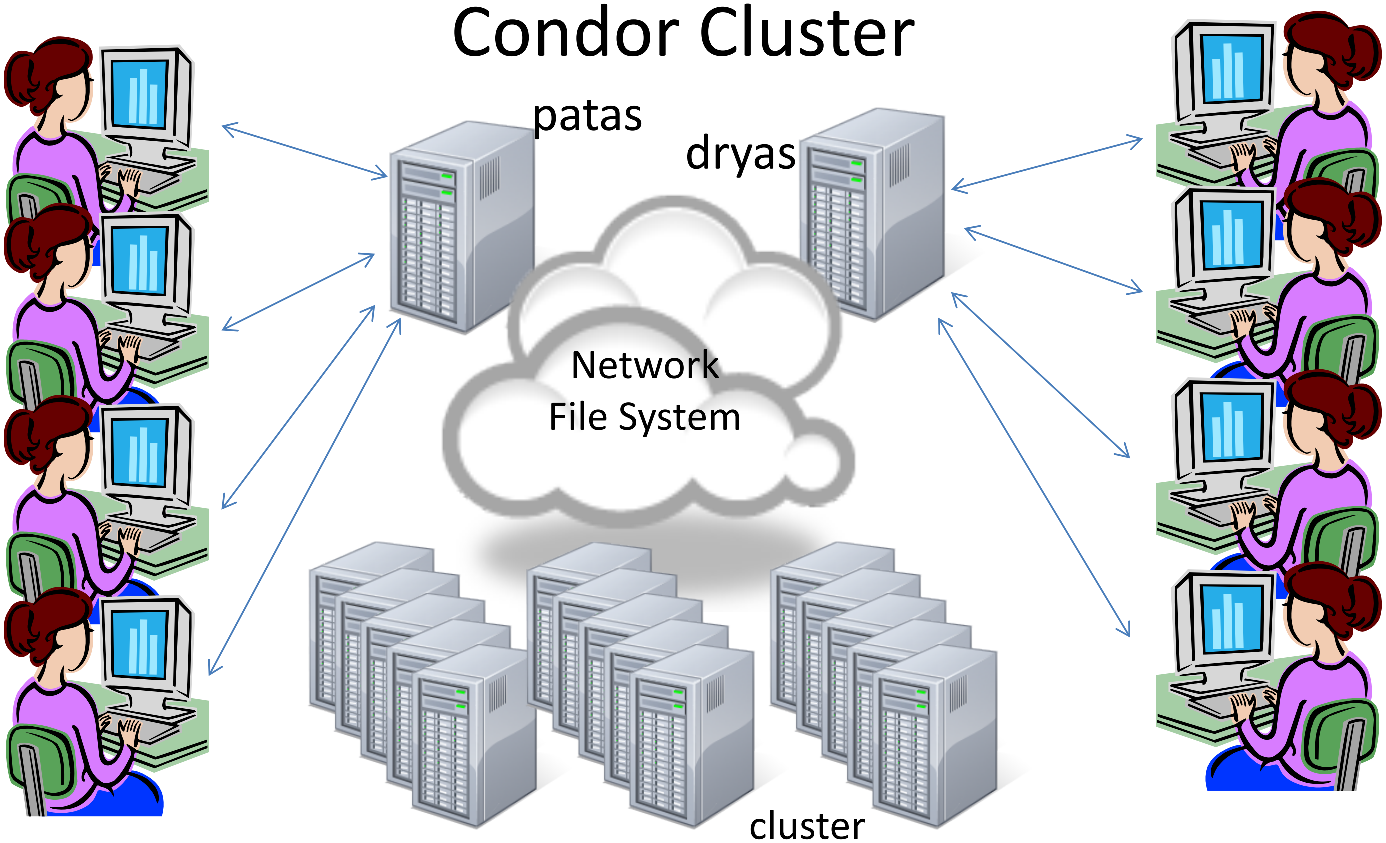
Primitive tokenization

```
$ cat moby_dick.html |            # echo the text
tr [:upper:] [:lower:] |          # convert to lower case
tr ' ' '\n' |                     # put each word on a line
grep -v ^$ |                      # get rid of blank lines
grep -v '<' |                     # get rid of HTML tags
grep -o "[a-z']*" |               # only want letters and '
sort |                            # sort the words
uniq |                            # find the vocabulary
wc -l                             # count them
   3956
```

If you are going to attempt to do project 1 using only unix shell utilities, you might need something like this

Review Project 1

Condor Cluster



Condor

```
$ condor_submit myjob.cmd
```

```
universe            = vanilla
executable          = /usr/bin/python
getenv              = true
input               = myinput.in
output              = myoutput.out
error               = myerror.err
log                 = mylogfile.log
arguments           = "myprogram.py -x"
transfer_executable = false
queue
```

The system will send you email when your job is complete.

Using variables in Condor files

`flexible.job`

```
file_ext            = $(depth)_$(gain)
universe            = vanilla
executable          = /opt/mono/bin/mono
getenv              = true
output              = acc_file.$(file_ext)
error               = q4.err
log                 = /tmp/davinman/q4.log
arguments           = “myprog.exe model_file.$(file_ext) sys_file.$(file_ext)"
transfer_executable = false
queue
```

```
$ condor_submit -append "depth=20” -append "gain=4" flexible.job
```

Project 1

- Due August 3, 11:45 p.m.
- Counting syntactic constituents in a corpus
- You may use regular expressions for this
  - (not a requirement if you prefer other methods)
- Using Linux
- Using the Condor system
- Packaging and submitting assignments