LING/C SC/PSYC 438/538

Lecture 13

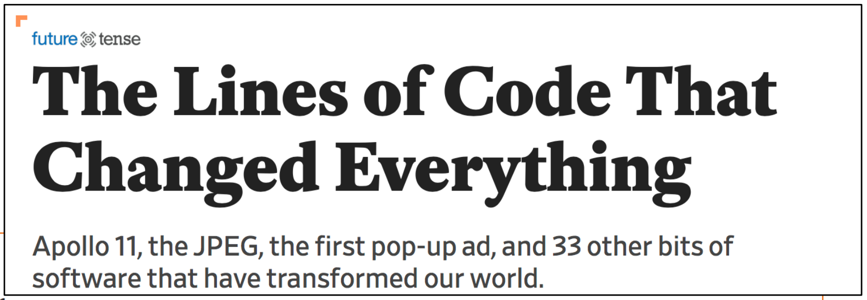
Sandiway Fong

Today's Topics

- An interesting read
- More on Perl regex:
    - a note on exponential time matching
    - Perl code inside a regex!
    - Character and word frequency counting
- Zipf's Law
    - Brown Corpus case study

Interesting read

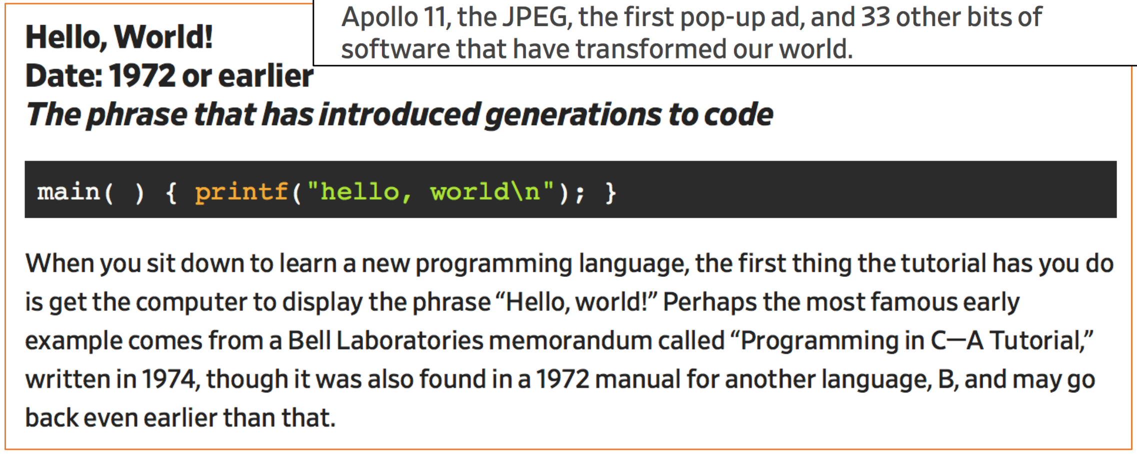
- https://slate.com/technology/2019/10/consequential-computer-code-software-history.html



future tense

The Lines of Code That Changed Everything

Apollo 11, the JPEG, the first pop-up ad, and 33 other bits of software that have transformed our world.

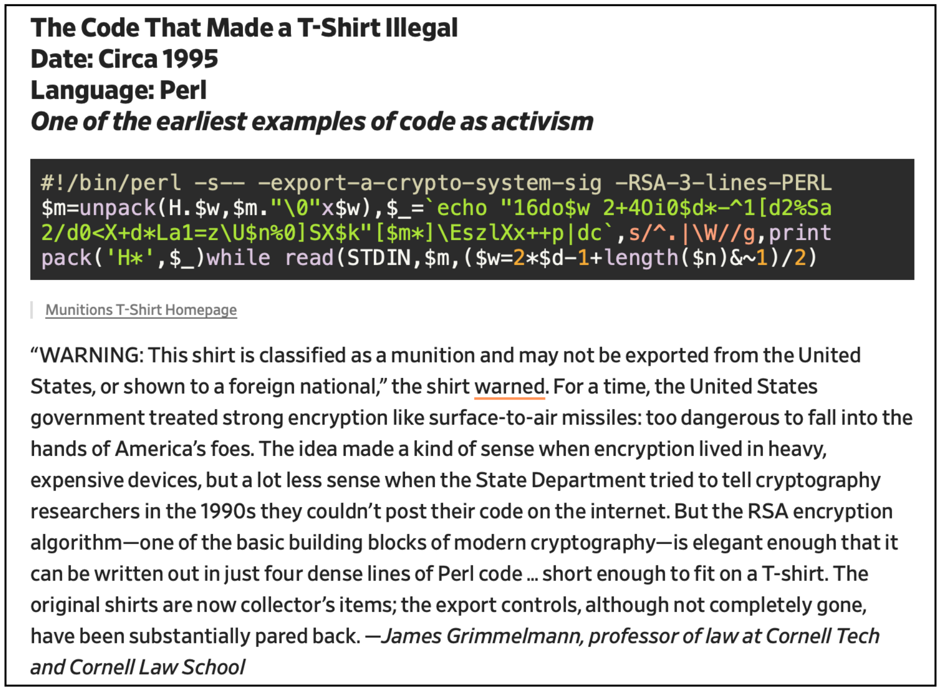


Hello, World!
Date: 1972 or earlier
The phrase that has introduced generations to code

```
main( ) { printf("hello, world\n"); }
```

When you sit down to learn a new programming language, the first thing the tutorial has you do is get the computer to display the phrase "Hello, world!" Perhaps the most famous early example comes from a Bell Laboratories memorandum called "Programming in C—A Tutorial," written in 1974, though it was also found in a 1972 manual for another language, B, and may go back even earlier than that.

Perl code and History



The Code That Made a T-Shirt Illegal
Date: Circa 1995
Language: Perl
One of the earliest examples of code as activism

```
#!/bin/perl -s-- -export-a-crypto-system-sig -RSA-3-lines-PERL
$m=unpack(H.$w,$m."\0"x$w),$_=`echo "16do$w 2+4Oi0$d*-^1[d2%Sa
2/d0<X+d*La1=z\U$n%0]SX$k"[$m*]\EszlXx++p|dc`,s/^.|\W//g,print
pack('H*',$_)while read(STDIN,$m,($w=2*$d-1+length($n)&~1)/2)
```

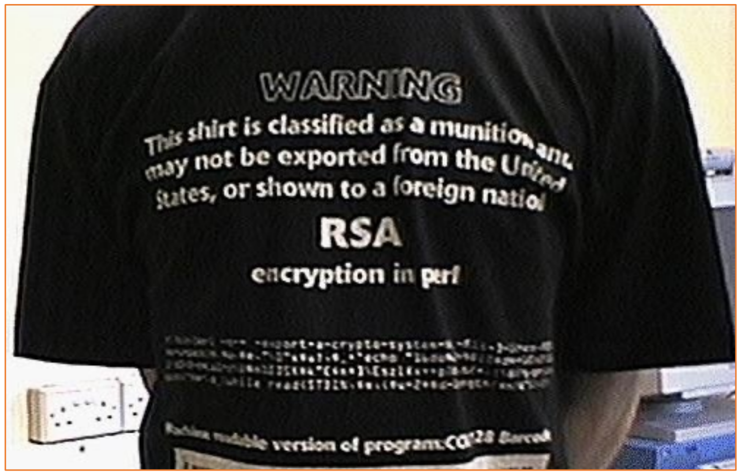
Munitions T-Shirt Homepage

"WARNING: This shirt is classified as a munition and may not be exported from the United States, or shown to a foreign national," the shirt warned. For a time, the United States government treated strong encryption like surface-to-air missiles: too dangerous to fall into the hands of America's foes. The idea made a kind of sense when encryption lived in heavy, expensive devices, but a lot less sense when the State Department tried to tell cryptography researchers in the 1990s they couldn't post their code on the internet. But the RSA encryption algorithm—one of the basic building blocks of modern cryptography—is elegant enough that it can be written out in just four dense lines of Perl code … short enough to fit on a T-shirt. The original shirts are now collector's items; the export controls, although not completely gone, have been substantially pared back. —*James Grimmelmann, professor of law at Cornell Tech and Cornell Law School*

Perl code and History

http://www.cypherspace.org/adam/rsa/rsa-details.html

```
#!/bin/perl -sp0777i<X+d*lMLa^*lN%0]dsXx++lMlN/dsM0<j]dsj
$/=unpack('H*',$_);$_=`echo 16dio\U$k"SK$/SM$n\EsN0p[lN*1
lK[d2%Sa2/d0$^Ixp"|dc`;s/\W//g;$_=pack('H*',/((..)*)$/)
```

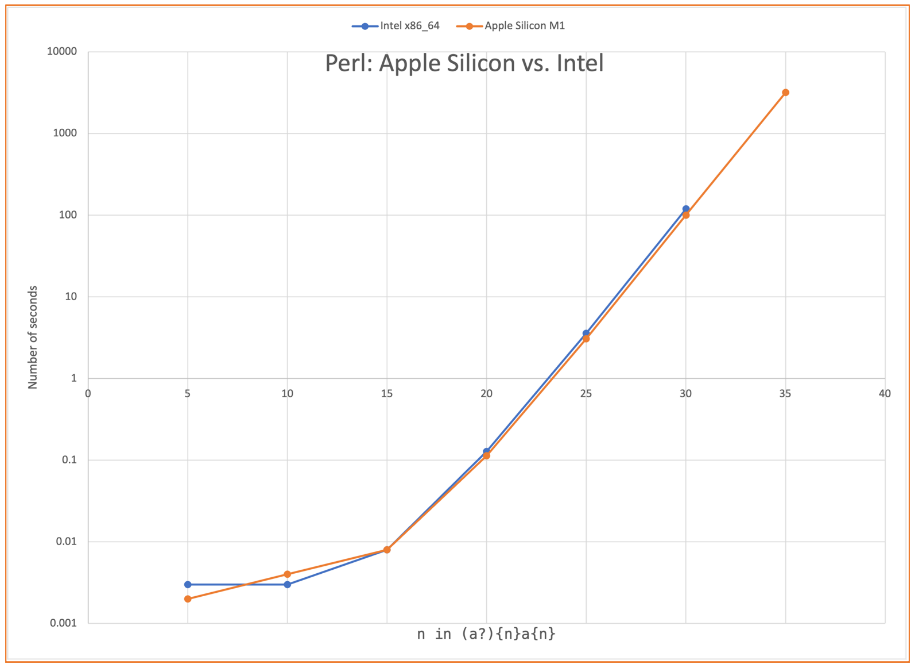

WARNING
This shirt is classified as a munition and may not be exported from the United States, or shown to a foreign national
RSA
encryption in perl

Last Time: exponential time regex worst case

- Consider /a?a?a?aaa/ matching against the string *aaa*
- Below, blue a = match, green a = skip, red a = fail:

1. a?a?a?aaa
2. a?a?a?aaa
3. a?a?a?aaa
4. a?a?a?aaa
5. a?a?a?aaa
6. a?a?a?aaa
7. a?a?a?aaa
8. a?a?a?aaa

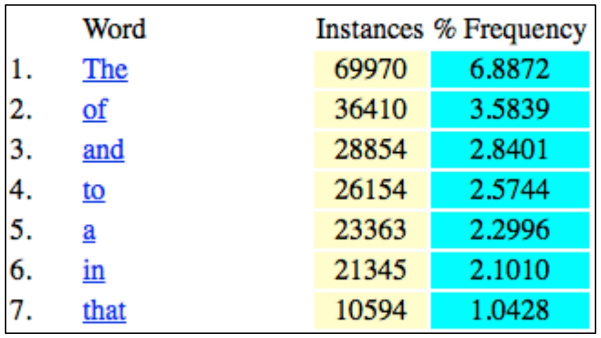
Perl implementations



- perl 5, version 28, subversion 3 (v5.28.3) built for darwin-thread-multi-2level
  - /opt/local/bin/perl: Mach-O 64-bit executable x86_64
- perl 5, version 34, subversion 1 (v5.34.1) built for darwin-thread-multi-2level
  - /opt/local/bin/perl: Mach-O 64-bit executable arm64

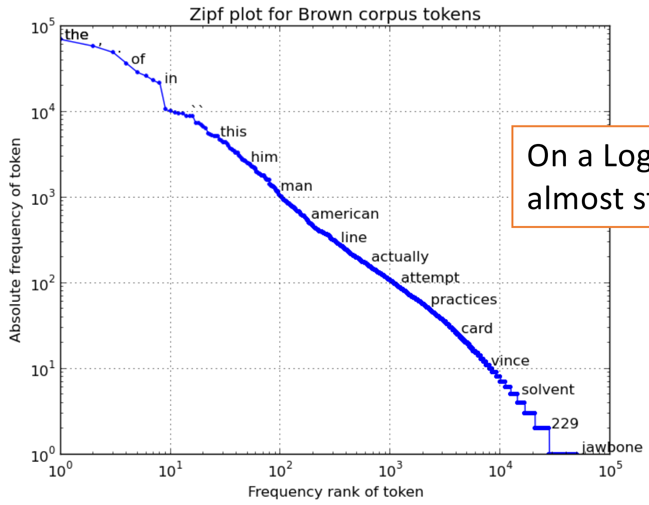
Zipf's Law

- Zipf's law states that given some corpus of natural language utterances, the frequency of any word is inversely proportional to its rank in the frequency table.
- See:
  - http://demonstrations.wolfram.com/ZipfsLawAppliedToWordAndLetterFrequencies/
  - Brown Corpus (1,015,945 words): only 135 words are needed to account for half the corpus.



	Word	Instances	% Frequency
1.	The	69970	6.8872
2.	of	36410	3.5839
3.	and	28854	2.8401
4.	to	26154	2.5744
5.	a	23363	2.2996
6.	in	21345	2.1010
7.	that	10594	1.0428

http://www.learnholistically.it/esp-clil/wfk2.htm

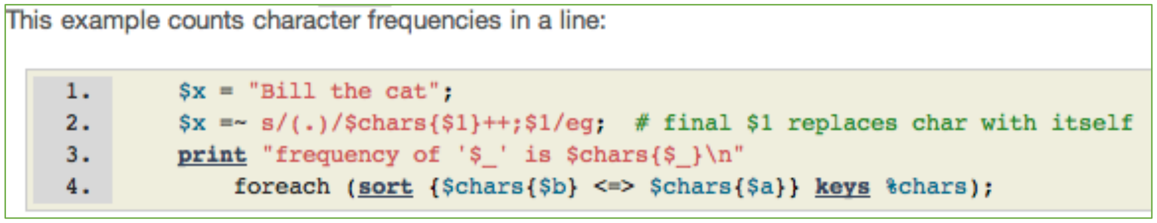


On a Log – Log scale: almost straight line

https://finnaarupnielsen.wordpress.com/2013/10/22/zipf-plot-for-word-counts-in-brown-corpus/

Character Frequency Counting

- Sample code is rather interesting (e flag):



This example counts character frequencies in a line:

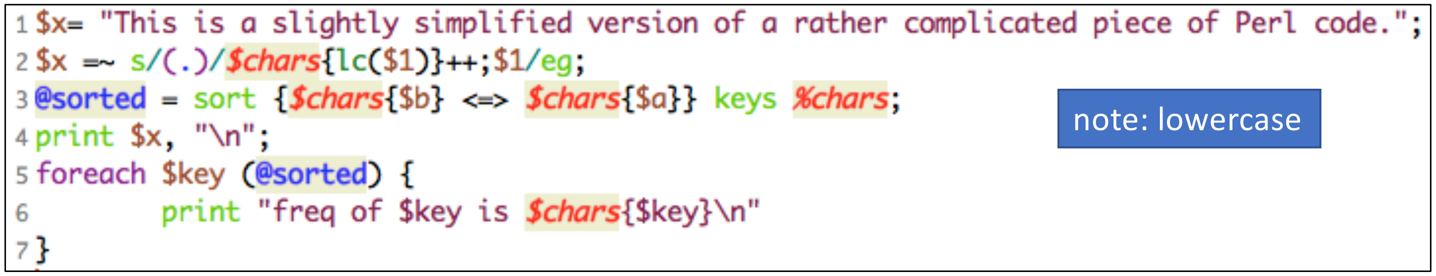
```
$x = "Bill the cat";
$x =~ s/(.)/$chars{$1}++;$1/eg;  # final $1 replaces char with itself
print "frequency of '$_' is $chars{$_}\n"
    foreach (sort {$chars{$b} <=> $chars{$a}} keys %chars);
```

`s/regex/replace/e`

- Generally, in regex:

`(?{ Perl code })`

- Slightly modified but easier to read:



```
$x= "This is a slightly simplified version of a rather complicated piece of Perl code.";
$x =~ s/(.)/$chars{lc($1)}++;$1/eg;
@sorted = sort {$chars{$b} <=> $chars{$a}} keys %chars;
print $x, "\n";
foreach $key (@sorted) {
        print "freq of $key is $chars{$key}\n"
}
```

note: lowercase

Character Frequency Counting

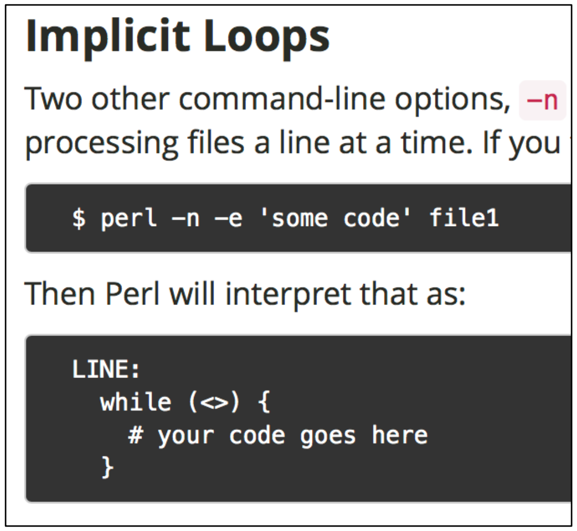
```
$ perl -le '$l = qq/@ARGV/; $l =~ s/(.)/$c{$1}++/eg; foreach $k (sort {$c{$b} <=>
$c{$a}} keys %c) {print "$k:$c{$k} "} ' this is a sentence to test this program.
 :7
t:6
s:5
e:4
i:3
r:2
h:2
a:2
n:2
o:2
g:1
m:1
.:1
p:1
c:1
```

Steps:

1. `$l = qq/@ARGV/` put the command arguments into a string `$l`
2. globally match each character `(.)` and increment a hash table (`%c`) count
3. for each key `$k` in `%c` (sorted in descending order by value), print key:value

Character Frequency Counting

```
perl -ne '{s/(.)/$c{lc($1)}++/eg} END {foreach $k (sort {$c{$b} <=> $c{$a}} keys %c) {print "$k:$c{$k} "}
print "\n"}' 12thnight.txt
```



Implicit Loops

Two other command-line options, `-n`
processing files a line at a time. If you

```
$ perl -n -e 'some code' file1
```

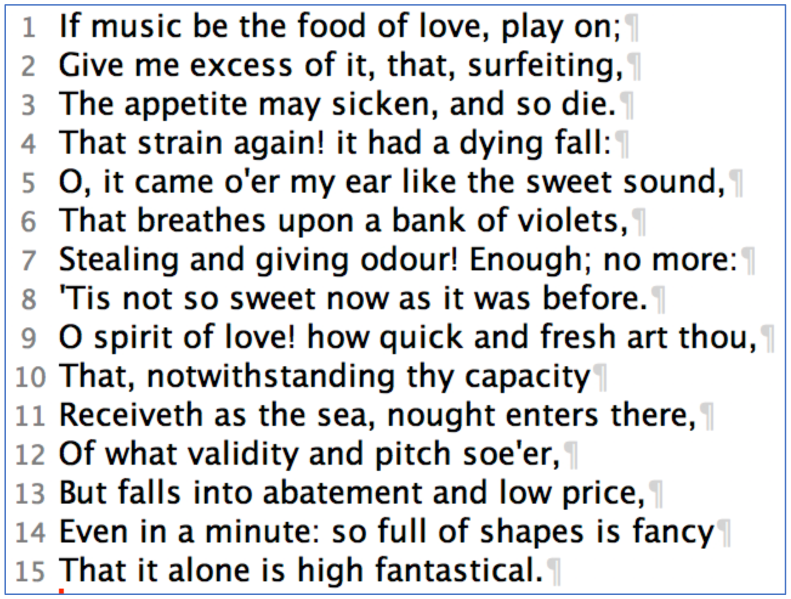
Then Perl will interpret that as:

```
LINE:
  while (<>) {
    # your code goes here
  }
```

Note:

- $_ will contain each line of `file1`
- *loop code* END *post-loop code*

12thnight.txt



```
If music be the food of love, play on;
Give me excess of it, that, surfeiting,
The appetite may sicken, and so die.
That strain again! it had a dying fall:
O, it came o'er my ear like the sweet sound,
That breathes upon a bank of violets,
Stealing and giving odour! Enough; no more:
'Tis not so sweet now as it was before.
O spirit of love! how quick and fresh art thou,
That, notwithstanding thy capacity
Receiveth as the sea, nought enters there,
Of what validity and pitch soe'er,
But falls into abatement and low price,
Even in a minute: so full of shapes is fancy
That it alone is high fantastical.
```

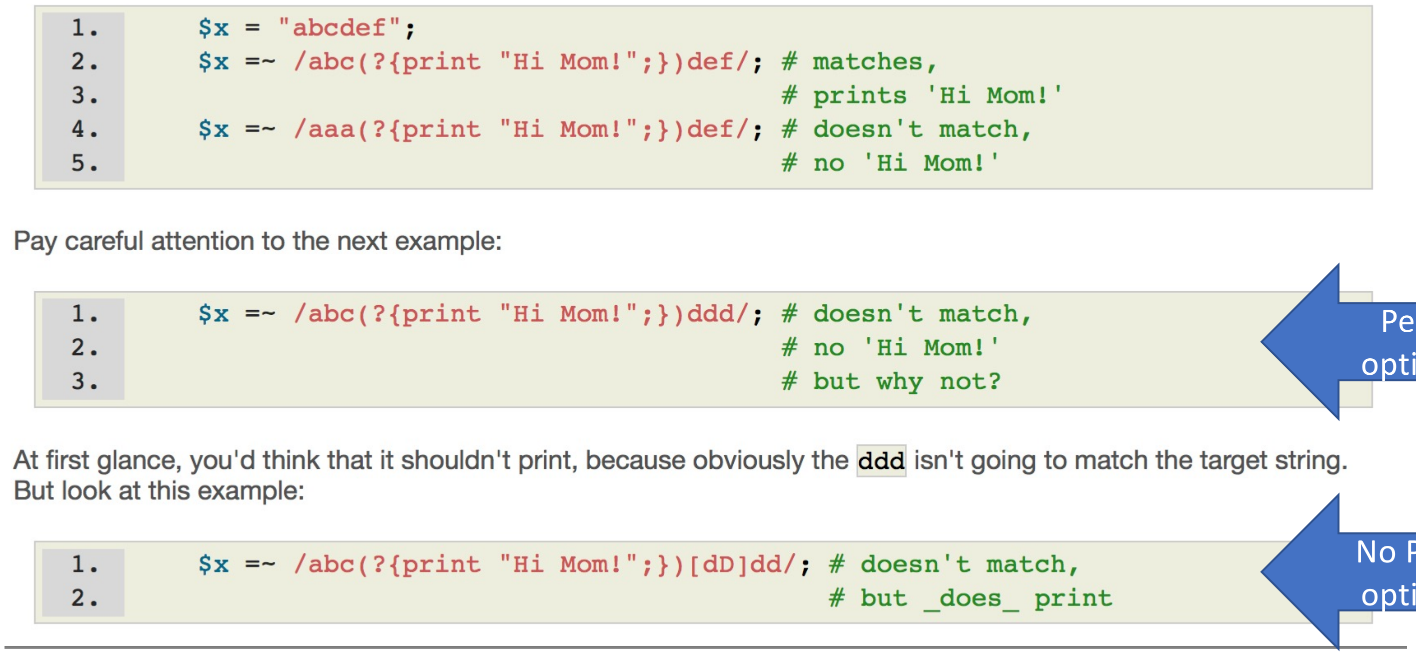
Character Frequency Counting

12th Night:

```
perl -ne '{s/(.)/$c{lc($1)}++/eg} END {foreach
$k (sort {$c{$b} <=> $c{$a}} keys %c) {print
"$k:$c{$k} "} print "\n"}' 12thnight.txt
```

```
 :99 t:52 e:51 a:45 i:40 o:36 n:33 s:31 h:25
f:16 l:16 r:16 ,:14 d:13 u:12 c:12 g:11 p:9 w:8
m:8 v:8 y:8 b:6 k:4 ':3 .:3 !:3 ::3 ;:2 x:1 q:1
```

(?{ Perl code })



```
    $x = "abcdef";
    $x =~ /abc(?{print "Hi Mom!";})def/; # matches,
                                         # prints 'Hi Mom!'
    $x =~ /aaa(?{print "Hi Mom!";})def/; # doesn't match,
                                         # no 'Hi Mom!'
```

Pay careful attention to the next example:

```
    $x =~ /abc(?{print "Hi Mom!";})ddd/; # doesn't match,
                                         # no 'Hi Mom!'
                                         # but why not?
```

Perl regex optimization

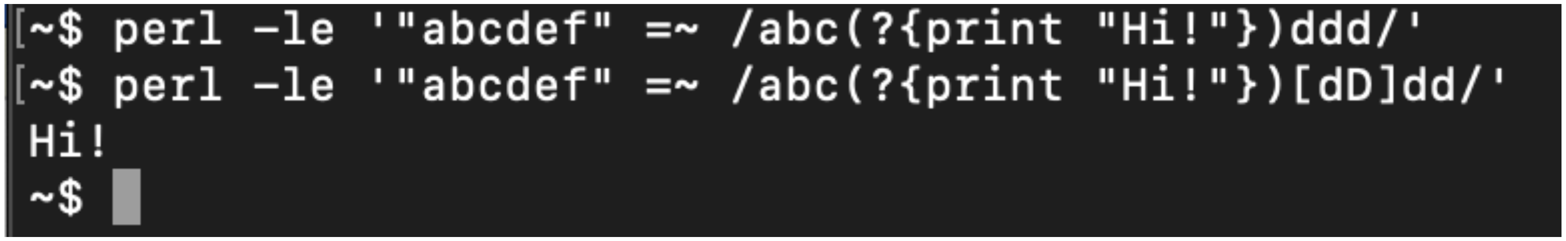
At first glance, you'd think that it shouldn't print, because obviously the `ddd` isn't going to match the target string. But look at this example:

```
    $x =~ /abc(?{print "Hi Mom!";})[dD]dd/; # doesn't match,
                                            # but _does_ print
```

No Perl regex optimization

(?{ Perl code })

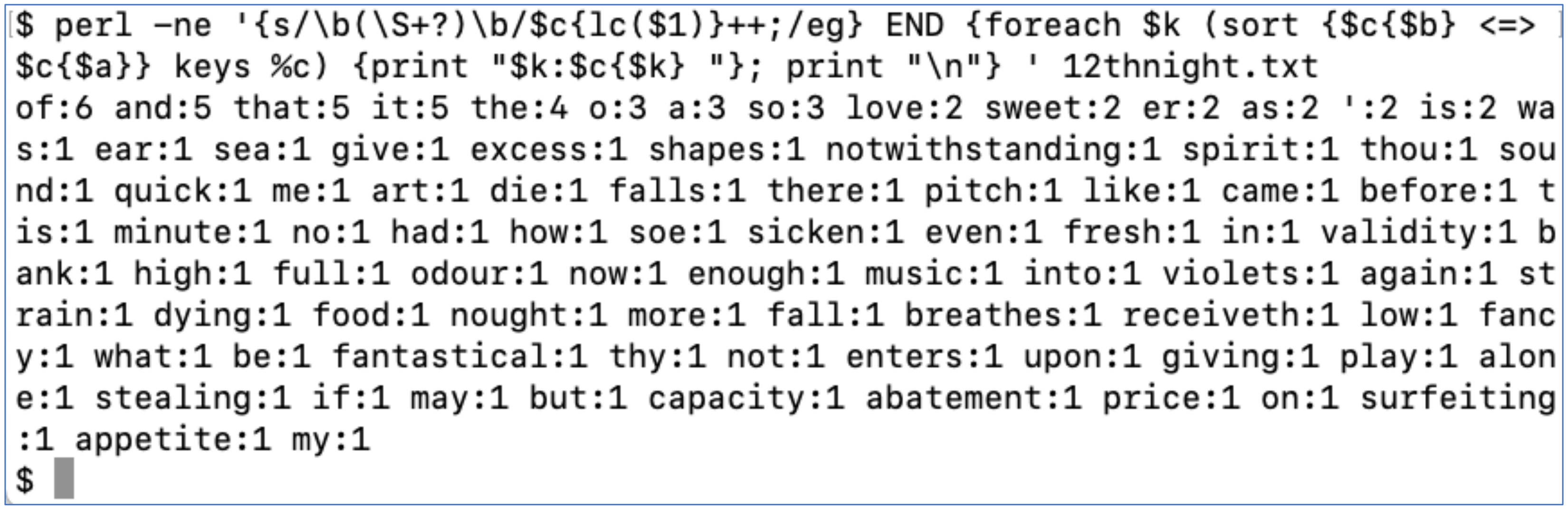
- One-liner:



```
[~$ perl -le '"abcdef" =~ /abc(?{print "Hi!"})ddd/'
[~$ perl -le '"abcdef" =~ /abc(?{print "Hi!"})[dD]dd/'
Hi!
~$
```

Word Frequency Counting

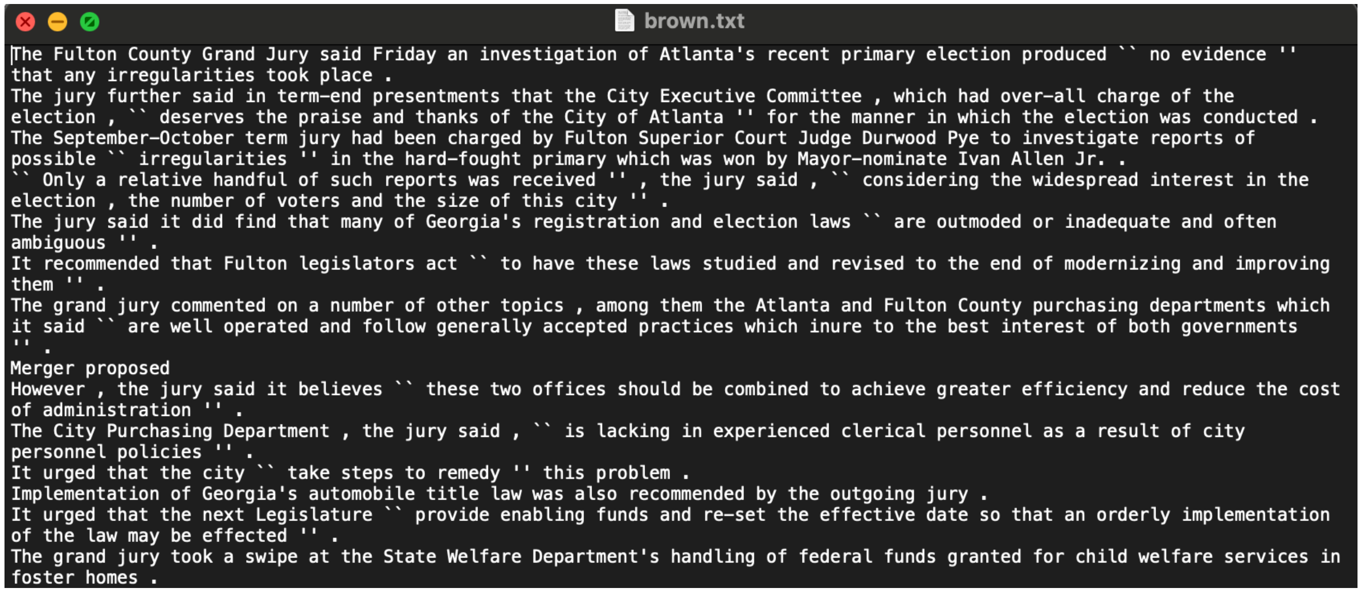
- Words, including punctuation: `\b(\S+?)\b`



```
$ perl -ne '{s/\b(\S+?)\b/$c{lc($1)}++;/eg} END {foreach $k (sort {$c{$b} <=> 
$c{$a}} keys %c) {print "$k:$c{$k} "}; print "\n"} ' 12thnight.txt
of:6 and:5 that:5 it:5 the:4 o:3 a:3 so:3 love:2 sweet:2 er:2 as:2 ':2 is:2 wa
s:1 ear:1 sea:1 give:1 excess:1 shapes:1 notwithstanding:1 spirit:1 thou:1 sou
nd:1 quick:1 me:1 art:1 die:1 falls:1 there:1 pitch:1 like:1 came:1 before:1 t
is:1 minute:1 no:1 had:1 how:1 soe:1 sicken:1 even:1 fresh:1 in:1 validity:1 b
ank:1 high:1 full:1 odour:1 now:1 enough:1 music:1 into:1 violets:1 again:1 st
rain:1 dying:1 food:1 nought:1 more:1 fall:1 breathes:1 receiveth:1 low:1 fanc
y:1 what:1 be:1 fantastical:1 thy:1 not:1 enters:1 upon:1 giving:1 play:1 alon
e:1 stealing:1 if:1 may:1 but:1 capacity:1 abatement:1 price:1 on:1 surfeiting
:1 appetite:1 my:1
$ 
```

Brown Corpus

- https://en.wikipedia.org/wiki/Brown_Corpus
  - "It contains 500 samples of English-language text, totaling roughly one million words, compiled from works published in the United States in 1961."

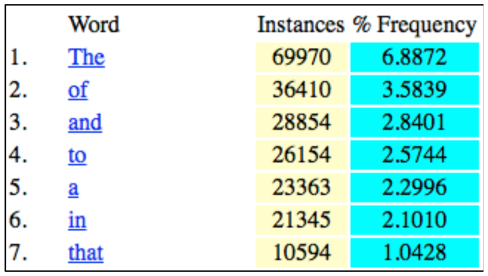


```
The Fulton County Grand Jury said Friday an investigation of Atlanta's recent primary election produced `` no evidence ''
that any irregularities took place .
The jury further said in term-end presentments that the City Executive Committee , which had over-all charge of the
election , `` deserves the praise and thanks of the City of Atlanta '' for the manner in which the election was conducted .
The September-October term jury had been charged by Fulton Superior Court Judge Durwood Pye to investigate reports of
possible `` irregularities '' in the hard-fought primary which was won by Mayor-nominate Ivan Allen Jr. .
`` Only a relative handful of such reports was received '' , the jury said , `` considering the widespread interest in the
election , the number of voters and the size of this city '' .
The jury said it did find that many of Georgia's registration and election laws `` are outmoded or inadequate and often
ambiguous '' .
It recommended that Fulton legislators act `` to have these laws studied and revised to the end of modernizing and improving
them '' .
The grand jury commented on a number of other topics , among them the Atlanta and Fulton County purchasing departments which
it said `` are well operated and follow generally accepted practices which inure to the best interest of both governments
'' .
Merger proposed
However , the jury said it believes `` these two offices should be combined to achieve greater efficiency and reduce the cost
of administration '' .
The City Purchasing Department , the jury said , `` is lacking in experienced clerical personnel as a result of city
personnel policies '' .
It urged that the city `` take steps to remedy '' this problem .
Implementation of Georgia's automobile title law was also recommended by the outgoing jury .
It urged that the next Legislature `` provide enabling funds and re-set the effective date so that an orderly implementation
of the law may be effected '' .
The grand jury took a swipe at the State Welfare Department's handling of federal funds granted for child welfare services in
foster homes .
```

Brown Corpus

- brown.txt (lines taken from brown.sents() from nltk.corpus)

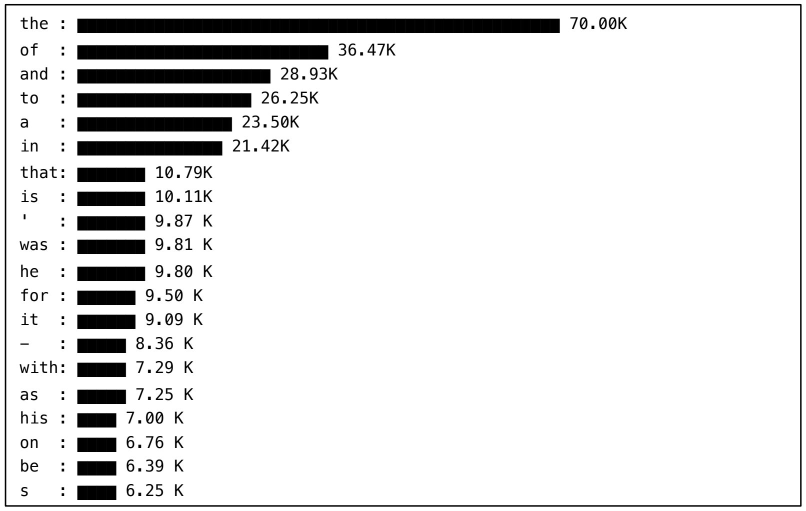
```
perl -ne '{s/\b(\S+?)\b/$c{lc($1)}++;/eg} END {@s = sort {$c{$b} <=>
$c{$a}} keys %c; foreach $k (@s[0..99]) { print "$k:$c{$k} " }}'
brown.txt

the:70003 of:36473 and:28935 to:26247 a:23502 in:21422 that:10789
is:10109 ':9865 was:9815 he:9801 for:9500 it:9094 -:8355 with:7290
as:7255 his:6999 on:6765 be:6388 s:6250 i:5932 at:5377 by:5346 this:5146
had:5133 not:4620 are:4394 but:4382 from:4371 or:4226 have:3942
they:3763 an:3751 you:3634 which:3561 one:3504 were:3285 all:3099
her:3036 she:2987 we:2844 there:2844 would:2719 their:2670 him:2619
been:2472 has:2437 when:2331 who:2280 will:2251 t:2246 more:2225 no:2219
if:2198 out:2167 so:2033 up:1974 what:1968 said:1961 can:1942 its:1858
about:1817 than:1796 into:1791 them:1790 only:1748 other:1714 time:1695
new:1646 some:1618 could:1602 these:1573 two:1516 may:1402 first:1389
then:1380 do:1375 man:1364 any:1344 like:1343 my:1319 now:1317 over:1307
such:1303 our:1253 .:1209 me:1185 even:1173 most:1162 made:1147
after:1077 also:1069 did:1044 many:1037 before:1016 must:1015 well:1006
af:1005 back:976 through:974
```



	Word	Instances	% Frequency
1.	The	69970	6.8872
2.	of	36410	3.5839
3.	and	28854	2.8401
4.	to	26154	2.5744
5.	a	23363	2.2996
6.	in	21345	2.1010
7.	that	10594	1.0428

Brown Corpus

- Using termgraph:
  - pip3 install termgraph (*Python*)
  - termgraph [datafile]   (*two columns*: #1 label, #2 value)
- `perl -ne '{s/\b(\S+?)\b/$c{lc($1)}++;/eg} END {@s = sort {$c{$b} <=> $c{$a}} keys %c; foreach $k (@s[0..19]) { print "$k $c{$k}\n" }}' brown.txt | termgraph`



```
the : ▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇ 70.00K
of  : ▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇ 36.47K
and : ▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇ 28.93K
to  : ▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇ 26.25K
a   : ▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇ 23.50K
in  : ▇▇▇▇▇▇▇▇▇▇▇▇▇▇▇ 21.42K
that: ▇▇▇▇▇▇▇ 10.79K
is  : ▇▇▇▇▇▇▇ 10.11K
'   : ▇▇▇▇▇▇▇ 9.87 K
was : ▇▇▇▇▇▇▇ 9.81 K
he  : ▇▇▇▇▇▇▇ 9.80 K
for : ▇▇▇▇▇▇ 9.50 K
it  : ▇▇▇▇▇▇ 9.09 K
-   : ▇▇▇▇▇ 8.36 K
with: ▇▇▇▇▇ 7.29 K
as  : ▇▇▇▇▇ 7.25 K
his : ▇▇▇▇ 7.00 K
on  : ▇▇▇▇ 6.76 K
be  : ▇▇▇▇ 6.39 K
s   : ▇▇▇▇ 6.25 K
```

Brown Corpus

from the earlier slide:

- http://demonstrations.wolfram.com/ZipfsLawAppliedToWordAndLetterFrequencies/
- Brown Corpus (1,015,945 words): only 135 words are needed to account for half the corpus.

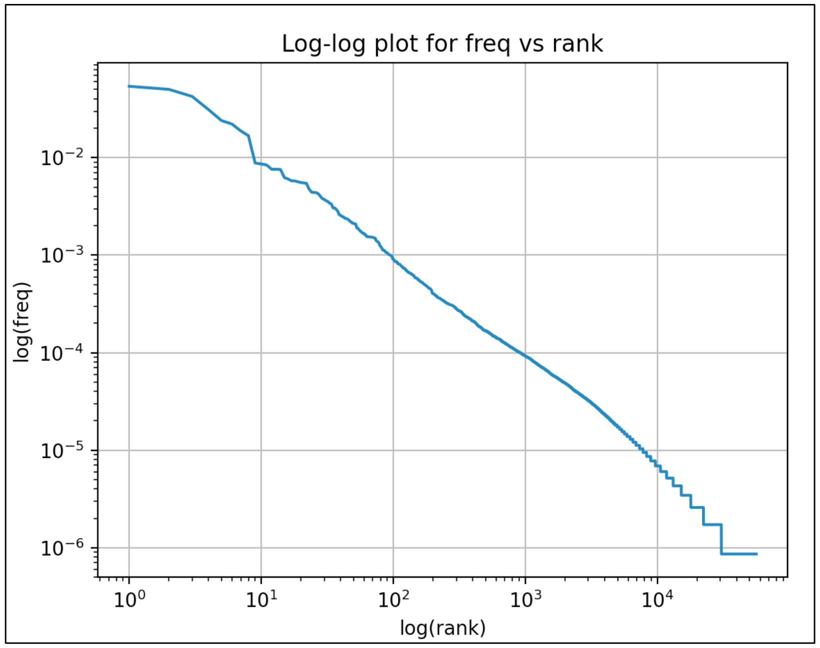
```
perl -ne '{s/\b(\S+?)\b/$c{lc($1)}++;$t++;/eg} END {$t /=2; foreach $k (sort
{$c{$b}<=>$c{$a}} keys %c) {$n++; $t -= $c{$k}; last if ($t<0) }; print "$n\n"}'
brown.txt
122
```

`last`	means terminate the running of foreach loop early
`$t`	total number of words, halved by `$t /= 2`
`$n`	to count the number of different words (`$n++`), stop when `$t<0`

Brown Corpus

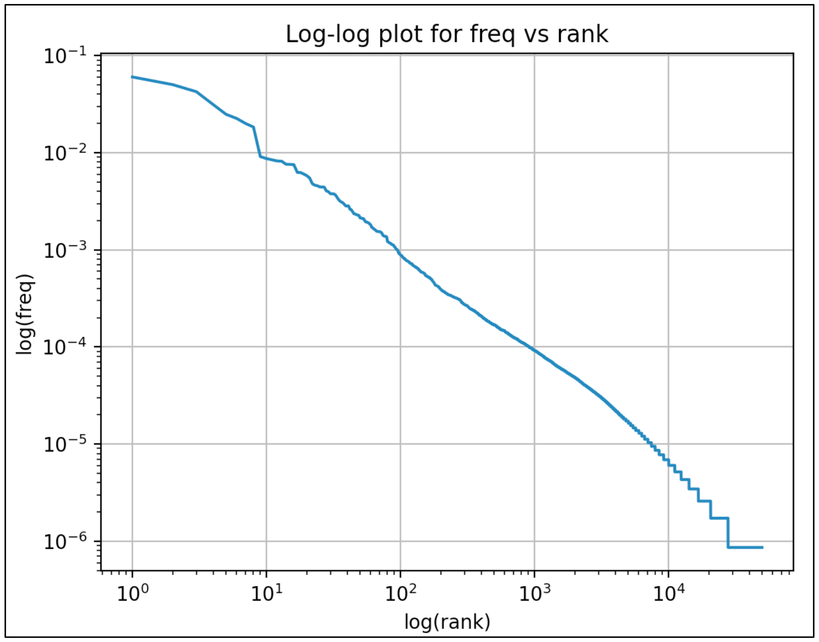
- In nltk (Natural Language Toolkit):

```
python –i zipf.py
>>> import nltk
>>> from nltk.corpus import
brown
>>> plot(brown.words())
```



Brown Corpus

```
plot([w.lower() for w in brown.words()])
```


Log-log plot for freq vs rank
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
log(freq)
10^{0}
10^{1}
10^{2}
10^{3}
10^{4}
log(rank)