Ctl160 Tekstikorpusten tietojenkäsittely 490160-0 Viides luento, 24.2.2003

Nicholas Volk

Yleisen kielitieteen laitos, Helsingin yliopisto

HELSINGIN YLIOPISTON
YLEISEN KIELITIETEEN LAITOS

`s/REGEXP/STRING/;` revisited...

- `s///`:n ensimmäinen osa on siis säännöllinen lauseke
- Säännöllisen lausekkeen ei tarvitse osua koko riviin, pelkkä osajono riittää
- Jos osutaan koko riviin, niin `g`-optiosta huolimatta osumia voi olla vain yksi
- Osuma-alue, ei sitä edeltäviä eikä seuraavia osia, korvataan annetulla merkkijonolla
- Merkkijono voi sisältää muuttujia (esim. `$1`)
- Merkkijonon merkeillä ei ole säännöllisten lausekkeiden mukaisia erikoismerkityksiä

Lisää Perlin säännöllisistä lausekkeista

- `$1` .. `$9` saivat siis arvon säännöllisessä lausekkeessa
- Ko. muuttujien arvoa ei voi itse muuttaa
- Niiden arvo on siis voimassa seuraavaan *osuvaan* säännölliseen lausekkeeseen asti
- Muuttujia `$`` ` ``, `$&` ja `$'` toimivat samalla periaatteella
- `$`` ` on osumaa edeltävä osa merkkijonosta
- `$&` on itse osuma, seuraavat tarkoittavat samaa:

```
perl -pe 's/[.]/ $&/g;
sed 's/[.]/ &/g'
```

- `$'` on osuman jälkeen tuleva osa

Esimerkki `osuma.perl`

```
#!/usr/bin/perl
while (<>) {
  /(a(.*)a)/;
  print "Ennen: $`\n";
  print "Osuma: $&\n";
  print "Loppu: $'\n";
  print "\$1: $1\n";
  print "\$2: $2\n";
  print "\$3: $3\n";
}
```

Säännöllisiä lausekkeita

- Olemme tutustuneet `s///`:n optiohin `g` ja `i`
- `s///`:ssä on rajoite: mm. `.*` ei oletusarvoisesti mene `\n`:n läpi...
- Tämä voi aiheuttaa yllätyksiä:

```
$ perl -e '$_="abc\ndef\"; s/a.*e//; print $_;'
abc
def
$
```

Lisää `s///`:n optiota

- Optio `s` tulkitsee merkkijonon yksiriviseksi, eli `\n` on merkki muiden joukossa
- Optio `m` on tavallaan sen vastakohta, merkkijono tulkitaan useita rivejä sisältäväksi, eli `^` ja `$` voivat löytyä monta kertaa.

```
$ perl -e '$_="abc\n"; s/^.*$//; print $_;'

$ perl -e '$_="abc\n"; s/^.*$//m; print $_;'

$ perl -e '$_="abc\n"; s/^.*$//s; print $_;'
$
```

Lisää esimerkkejä

```
$ perl -e '$_="abc\ndef\n"; s/a.*e//; print $_;'
abc
def
$ perl -e '$_="abc\ndef\n"; s/a.*e//m; print $_;'
abc
def
$ perl -e '$_="abc\ndef\n"; s/a.*e//s; print $_;'
f
$ # alusta loppuun:
$ perl -e '$_="abc\ndef\n"; s/^.*$//; print $_;'
abc
def
$ perl -e '$_="abc\ndef\n"; s/^.*$//m; print $_;'

def
$ perl -e '$_="abc\ndef\n"; s/^.*$//mg; print $_;'


$ perl -e '$_="abc\ndef\n"; s/^.*$//s; print $_;'
$
```

split(/LAUSEKE/, merkkijono)

- Käskyllä `split` voi jakaa merkkijonon, joka voi olla muuttuja, paloiksi
- Palojen erotin annetaan säännöllisenä lausekkeena
- Säännöllisen muotoinen syöte voidaan siis paloitella kerralla eri muuttujiin
- Alla näyte Susanne-korpuksesta:

```
G01:0010b    JJ      NORTHERN     northern     [O[S[Np:s.
G01:0010c    NN2     liberals     liberal      .Np:s]
G01:0010d    VBR     are          be           [Vab.Vab]
G01:0010e    AT      the          the          [Np:e.
G01:0010f    JB      chief        chief        .
G01:0010g    NN2     supporters   supporter    .
G01:0010h    IO      of           of           [Po.
G01:0010i    JJ      civil        civil        [Np.
G01:0010j    NN2     rights       right        .Np]
```

Susanne-korpuksen rakenne

```
G01:0010b    JJ      NORTHERN     northern    [O[S[Np:s.
```

- Kentät ovat sarkain-merkein (`\t`) erotetut
- `<http://www.ilc.pi.cnr.it/EAGLES96/synlex/node25.html>`:
- Field 1: text references
- Field 2: Part of speech tags (morfologinen tulkinta)
- Field 3: The text words (saneet)
- Field 4: Base form (lemma)
- Field 5: Syntactic annotation
- `[` viittaa jonkin syntaktisen kokonaisuuden alkuun, `]` vastaavasti loppuun

Paloitellaan...

- Skripti `susanne.perl` paloittelee syötteen ja tulostaa sen osat:

```
#!/usr/bin/perl
while(<>) {
    chop($_);
    ($ref, $pos, $word, $lemma, $syn) = split(/\t/, $_);
    print "REF:\t$ref\n";
    print "POS:\t$pos\n";
    print "WORD:\t$word\n";
    print "LEMMA:\t$lemma\n";
    print "SYNTAX:\t$syn\n\n";
}
```

- Skripti `susanne2.perl` tekee saman käyttäen taulukkoa...

Ongelmia paloittelussa

- Periaatteessa esimerkiksi sanat voisi paloitella käskyllä `split(/\s+/, $muuttuja)`
- Enemmän sanoja kuin "vastaanottavia"muuttujia: tietoa häviää
- Vähemmän sanoja kuin muuttujia: osalle muuttujista ei sijoiteta arvoa
- Tallettamalla sanat indeksitaulukkoon ongelma poistuu
- Kukin pala tallettuu taulukkoon omaan lokeroonsa eli alkioon

Connexorin FDG-jäsentimen tulostusta

- Connexorinkin jäsentimen antama tekstituloste on selkeä:
  1. Saneen sijainti lauseessa
  2. Sane
  3. Lemma
  4. Funktionaalinen dependenssi
  5. Pintasyntaktinen tagi ja morfologinen analyysi
- Kentät on erotettu sarkainmerkein
- Kenttä 4 voi olla myös tyhjä!
- Kenttä 5 voi olla moniselitteinen, jokaisella tulkinnalla oma kenttä!
- Eli kutakin sanetta kohden on 5+ kenttää! (Yleensä 5)

Indeksitaulukko

- Indeksitaulukkoon (kokonaisuutena) viitataan @-merkin avulla
- `split`-käsky voisi siis olla myös muotoa
  `@taulukko = split(/\t/, $muuttuja);`
- Taulukon yksittäiseen alkioon viitataan ideksin avulla, eli kertomalla monennestako alkiosta ollaan kiinnostuneita
- Indeksi kirjoitetaan hakasulkeiden perään eli taulukon ensimmäiseen alkioon viitattaisiin muuttujalla
  `$taulukko[0]`
- Eli tietojenkäsittelyteknistä syistä taulukko alkaa kohdasta 0, ei kohdasta!

Esimerkki

```
3       opetuksen       opetus  attr:>4 &A> N SG GEN
```

- Paloittelemalla yllä oleva rivi käskyllä
  `@taulukko = split(/\n/, $rivi)`
  saataisiin seuraavanlaiset arvot:
  - `$taulukko[0] = "3"`
  - `$taulukko[1] = "opetuksen"`
  - `$taulukko[2] = "opetus"`
  - `$taulukko[3] = "attr:>4"`
  - `$taulukko[4] = "&A> N SG GEN"`

Lisäasiaa taulukoista

- Näitä ei tarvitse osata itse käyttää, nämä vain näkyvät esimerkkiskripteissä, joten ne selitetään...
- Perlin komentoriviltä saamat argumentit tallettuvat parissa skriptissäkin nähtyyn `@ARGV`-taulukkoon
- Samassa yhteydessä nähty `$0` kertoo käytettävän ohjelman nimen
- `$#nimi` kertoo viimeisen alkion indeksin
- Koska alkiot alkavat nollasta, on `$#nimi` yhtä pienempi kuin taulukon koko

Lisäasiaa taulukoista (2)

- Numeerisessa yhteydessä `@nimi` kertoo taulukon koon, mutta ei-numeerisessa yhteydessä voi käydä hassusti...
- Käsky `shift @taulukko` palauttaa taulukon ensimmäisen (nollannen) alkion
- Samalla ko. alkio poistetaan taulukosta ja muiden alkioiden indeksiä pienennetään yhdellä
- Seuraavalla luennolla näytetään miten käydä komentoriviargumentit läpi shell-skriptissä, jos ehditään

KWIC-konkordanssi

- *Key Word In Context*
- Eli haluttu sane konteksteineen
- Perinteisessä korkordanssiohjelmassa ei kieliopillista älyä
- Lemmie-ohjelmalla (luento 1) pystyi hakemaan saneen sijasta lemmankin avulla...
- Kurssia varten tehty pieni korkordanssiohjelma `konko.perl`, joka kelpuuttaa saneen sijasta säännöllisen lausekkeen
- Laskareissa 2.1 ja 2.3 käytetyillä säännöllisillä lausekkeilla voisi yrittää matkia lemman avulla hakua...

Oktaaliluvut

- Kaikkia merkkejä ei aina pysty syöttämään näppäimistöltä
- Merkeillä on käytetyn merkkistandarin (mm. ISO-8859-1 ja Unicode) mukaiset numeeriset arvot
- Arvot esitetään yleensä oktaalilukuina (8-järjestelmä)
- Käskyn `tr ' ' '\012'` osa `\012` on siis oktaalikuku, joka tarkoittaa *Line Feed* -näppäintä
- Esim. `\007` on piip-ääni

Heksaluvut

- Perlissä voi viitata merkkiin myös heksalukuna (16-järjestelmä)
- Tällöin kenoviivan perään laitetaan `x` ja kaksinumeroinen heksaluku (`[0-9A-F]`):

```
$ perl -e '$a = <>; $a =~ tr/BA/\x41\102/; print $a;'
AB
BA
$
```

index ja rindex

- `index(STRING, SUBSTRING, INT)` kertoo monennestako merkistä *alusta* lukien haluttu osajono alkaa
- `index($_, ¨ ¨, 0)` etsisi siis alusta lukien ensimmäisen välilyönnin (sanarajan)
- Funktio palauttaa -1. jos osajonoa ei löydy
- `rindex` toimii muuten samoin, paitsi INT kertoo monennestako merkkijonon merkistä lähdetään etsimään välilyöntiä edeten kohti merkkijonon alkua
- `index` etsii siis vasemmalta oikealla
  `rindex` oikealta vasemmalle