11 Perl

Perl, (the *P*ractical *E*xtraction and *R*eporting *L*anguage), on ohjelmointikieli, jonka ensimmäisen version Larry Wall julkaisi vuonna 1987. Vuonna 1994 julkaistu versio 5 on saavuttanut jalansijaa etenkin WWW:n CGI-skriptien toteutustyökaluna. Vaikka Perliä ei toimitetakaan kaikkien järjestelmien mukana, on Perl saavuttanut jo “standardiaseman” Unix-työkalujen joukossa. Unix-ympäristöjen lisäksi Perlistä löytyy “portteja” ja versioita lähes kaikkiin järjestelmiin.

Perl on ohjelmointikielenä niin suuri kokonaisuus, ettei sen kaikkia ominaisuuksia voi oppia tai opettaa yhden luennon puitteissa. Tämän luvun tarkoituksena on käydä läpi Perlin perusteita ja perusrakenteita sekä antaa jonkinmoinen pohja Perl-koodin ymmärtämiseen.

11.1 Yleistä

11.1.1 Ominaisuuksia

Perliä voidaan käyttää (kuten `sed`iä ja `awk`ia) suodattamaan ja formatoimaan rivipohjaista (merkki)tietoa. Perl onkin (jo nimensäkin mukaan) erinomainen työkalu luotaessa selkeitä raportteja eri tavoin jäsennellystä merkkitiedosta. Perl sisältää näiden ominaisuuksien lisäksi kuitenkin suuren joukon ominaisuuksia, joita esimerkiksi `awk` ei tarjoa. Vahvoja Perlin ominaisuuksia ovat mm.

- kattavat ja tehokkaat kontrollirakenteet,
- kattavat funktiot tiedostojen käsittelyyn,
- toisten (“ulkoisten”) ohjelmien suoritus ja niiden tulosteiden suora käsittelymahdollisuus,
- tietokantojen käyttö,
- koneiden välinen kommunikointi (soketit),
- signaalien käsittely,
- sisäänrakennettu symbolinen debuggeri sekä
- moduulien käyttö.

11.1.2 Dokumentaatio

Perliin liittyvää tietoa ja dokumentaatiota on saatavilla runsaasti. Perlin mukana toimitetaan runsas dokumentaatio, joka Unix-järjestelmissä koostuu useista `man`-sivuista. "Sisällysluettelon" Perlin `man`-sivuista saa näkyviin itse Perlin päämanuaalista komennolla `man perl`. Lisäksi käytössä on komento `perldoc`, jolla saadaan selattua Perliin liittyvää dokumentaatiota eri hakusanoilla. Kattavan lista muista Perl-tietolähteistä löytyy Perl Instituutin WWW-sivuilta osoitteesta `http://www.perl.org`.

11.1.3 Perl-tulkki ja skriptin rakenne

Itse Perl-tulkki (`perl`) sijaitsee useimmissa järjestelmissä hakemistojen `/usr/bin/` tai `/usr/local/bin` alla. Jos järjestelmän mukana on toimitettu vanha Perlin versio (esim. versio 4), voivat uudemmat versiot sijaita kuitenkin jossain muualla (esim. `/opt` -hakemiston alla).

(Käyttäjän polussa sijaitsevan) Perlin version saa selville komennolla:

```
perl -v
```

Tällä hetkellä (Lokakuu 1999) viimeisin vakaa (stable) versio on 5.005_03.

Perl osaa lukea komentoja *stdin*ista ja lyhyitä (yksittäisiä) komentoja voidaan antaa käyttäen `-e` -optiota, mutta yleensä skripti kirjoitetaan suoritettavaksi seuraavan esimerkin mukaisesti.

```
#!/usr/local/bin/perl
print "Hello World\n";
```

Komennot ja lausekkeet eritellään toisistaan *puolipisteen* (`;`) avulla kuten C-kielessä.

11.2 Muuttujat

Aivan kuten `awk`issa, myös Perlissä muuttujat ovat "tyypittömiä". Käytännössä muuttujat ovat joko merkkijonoja (string) tai lukuja (vrt. int, float). Muuttujan käyttäytyminen esim. laskutoimituksissa määräytyy sen sisällön perusteella.

"Tavallinen" muuttuja Perlissä on muotoa: `$muuttujan_nimi`. Muuttujan nimi voi sisältää *alfanumeerisia* merkkejä sekä alaviivoja (_). Nimen maksimipituus on 255 merkkiä. Isoilla ja pienillä merkeillä on eri merkitys, eli esim. `$aBc` ja `$Abc` ovat eri muuttujia.

11.2.1 Merkkijonot

Merkkijonoa voidaan Perlissä ilmaista kahdella tavalla:

- yksinkertaisin lainausmerkein, `'merkkijono'` tai
- kaksinkertaisin lainausmerkein, `"merkkijono"`.

Kuten useimmissa shelleissä, tässäkin tapauksessa yksinkertaisten lainausmerkkien käyttö on "suojaavaa". Ts. yksinkertaisten lainausmerkkien sisällä ei tehdä korvauksia esimerkiksi muuttujille. Kaksinkertaisten lainausmerkkien sisällä puolestaan mm. muuttujaviittaukset korvataan muuttujien arvoilla. Myös *kenoviiva* "\" toimii suojaavana merkkinä (esim. `'`-merkin suojaus).

Esimerkki:

```
#!/usr/local/bin/perl
$a="foo"; $b='bar'; # tässä ei väliä
print '$a$b\n';     # tulostaa $a$b\n
print "$a$b\n";     # tulostaa foobar ja rivinvaihdon
```

11.2.2 Numeeriset arvot

Perl tuntee ja hyväksyy numeeriset arvot useassa eri muodossa. Luku voidaan ilmaista usealla eri tavalla. Se voidaan antaa kokonaislukuna (esim. `101`), desimaalilukuna (esim. `3.14159`) tai exponenttimuodossa (esim. `2.1E3`). Eksponenttimuodossa `E` voi olla myös `e` ja eksponenttiosa voi luonnollisesti olla negatiivinen (pienet luvut). Lisäksi luvun "hahmottamista" voidaan parantaa lisäämällä alaviivoja (_) numeroiden väliin (esim. `100_000` on sama kun `100000`).

11.2.3 Taulukot

Perlin taulukko-muuttujat ovat muotoa `@taulukon_nimi`. Perl tarjoaa joukon *taulukko-operaattoreita*, joilla taulukoiden käsittely hoidetaan.

Taulukoiden alustuksessa voidaan käyttää erityistä `qw`-operaattoria (*quoting operator*) tai kaarisulkuja "`()`". Esim.

```
@eka=("foo", "bar", 1, 2); # neljän alkion taulu
@toka=qw/foo bar 1 2/;     # käyttäen qw-operaattoria
```

Käytettäessä kaarisulkuja, kaikki merkkijonoalkiot on siis laitettava lainausmerkkien väliin. Operaattorin `qw` etuna on se, ettei alkioita tarvitse erikseen laittaa lainauksien väliin, vaan se tehdään automaattisesti. "Pysäytysmerkkinä" `qw`-operaattorissa edellä käytetty "`/`"-merkki voi olla (lähes) mikä tahansa "arvoon" sisältymätön merkki (vrt. `sed`).

Taulukon muodostamisessa voidaan käyttää myös `..`-operaattoria. Tämä "arvoväli" (range) -operaattori mahdollistaa viittauksen tiettyyn järjestettyyn alkioväliin (yleensä kirjaimia tai numeroita) lyhyemmällä tavalla. Esim.

```
@eka=(1 .. 10); # 10 alkiota, alkioina numerot 1-10
@toka=('a' .. 'j'); # 10 kpl, alkioina kirjaimet a-j
```

11.2.4 Taulukon indeksit

Edellä ei otettu vielä kantaa taulukon *indekseihin*. Tiettyihin (edellä esitetyllä tavalla alustettuihin) taulukon alkioihin voidaan viitata kuten C-kielessä, indeksinumerolla. Taulukon indeksointi alkaa nollasta (`0`). Nyt on huomattava, että viitatessa yhteen alkioon on `@`-merkki vaihdettava `$`-merkkiin. Kyseessä on siis itse asiassa taulukkoon `@taulu` liittyvä *muuttuja*. Esim.

```
@eka=(1 .. 10);     # 10 alkiota, alkioina numerot 1-10
@toka=('a' .. 'j'); # 10 kpl, alkioina kirjaimet a-j
$a=$eka[2];         # a saa arvon 3
$b=$toka[9];        # b saa arvon j
```

Jokaiseen taulukkoon liittyy myös toinen muuttuja, `$#taulu`, joka ilmaisee taulukon `@taulu` viimeisen indeksin arvon. Taulukon alkioiden lukumäärä on siis `$#taulu + 1`. Kuten edeltävästä on käynyt ilmi, ovat Perlin taulukot dynaamisia. Taulukon kokoa voidaan kasvattaa kasvattamalla muuttujan `$#taulu` arvoa. Toisaalta taulukko voidaan tyhjentää antamalla eo. muuttujalle arvoksi `-1`. Taulukkoon voidaan myös lisätä alkioita yksinkertaisesti ottamalla uusi indeksi käyttöön. Esim.

```
@taulu=qw/foo bar/ ; # 2 alkiota
```

```
$taulu[2]="foobar" ; # 3 alkiota ("foo","bar","foobar")
$#taulu=3;           # 4 alkiota ("foo","bar","foobar",undef)
$#taulu=-1;          # taulun tyhjennys
```

11.2.5 Assosiatiiviset taulukot

Perinteisten "numeroindeksitaulukoiden" lisäksi Perlissä voidaan käyttää *assosiatiivisia taulukoita*. Tämä tarkoittaa, että taulukossa käytetty indeksi voi olla arvoltaan mikä tahansa, esim. merkkijono.

Assosiatiiviset taulukot erotellaan "tavallisista" taulukoista käyttämällä `@`-merkin sijaan `%`-merkkiä. Assosiatiivinen taulukko on siis muotoa `%taulu`. Assosiatiivinen taulukko voidaan alustaa kuten tavallinenkin, mutta joka toinen alkio esittää seuraavaa alkiota vastaava indeksiä. Tiettyä alkiota vastaavaan indeksiin puolestaan viitataan antamalla indeksi hakasulkujen "`[]`" sijaan kaarisulkujen "`{}`" sisään. Esim.

```
%taulu=qw/eka foo toka bar/;       # 2 alkiota foo ja bar
print "$taulu{eka}$taulu{toka}\n"; # tulostaa foobar
```

11.2.6 Pinot ja jonot

Mainittakoon, että Perlissä taulukoita voidaan käsitellä myös *pinoina* ja *jonoina*. Käytettävissä on tarkoitukseen sopivat funktiot `push`, `pop`, `shift` ja `unshift`. Tarkempi käsittely tässä sivuutetaan.

11.2.7 Sisäisiä muuttujia

Seuraavassa muutamia Perlin sisäisiä muuttujia.

`$_`	Luettu rivi ("työmuisti"), oletusmuuttujana monelle komennolle
`@ARGV`	Taulukko sisältäen komentoriviargumentit.
`%ENV`	Assosiatiivinen taulukko ympäristömuuttujista.
`$.`	Luetun rivin numero.
`$/`	Tietue-erotin luettaessa, oletuksena rivinvaihto.
`@_`	Oman aliohjelman parametrilista.

11.3 Operaattorit

Perl sisältää suuren joukon operaattoreita muuttujien ja *skalaarien* käsittelyyn ja vertailuun. Tässä on esitelty ainoastaan tärkeimmät perusoperaattorit. Kattava dokumentaatio Perlin operaattoreista löytyy manuaalisivulta `perlop` (komento `man perlop` tai `perldoc perlop`).

11.3.1 Aritmeettiset operaattorit

Aritmeettisiin operaatioihin Perl tarjoaa tutut operaattorit.

- `+` *summa*, esim. `$a=5; $b=$a+5; # b=10`
- `-` *erotus*, esim. `$a=5; $b=$a-5; # b=0`
- `*` *tulo*, esim. `$a=5; $b=$a*5; # b=25`
- `/` *osamäärä*, esim. `$a=5; $b=$a/2; # b=2.5`
- `**` *potenssiin korotus*, esim. `$a=5; $b=$a**5; # b=3125`
- `%` *modulo* eli jakojäännös, esim. `$a=5; $b=$a%3; # b=2`

On hyvä muistaa, että potenssiin korotus on muista edellisistä poiketen *oikealle assosiatiivinen*. Ts. `2**2**3` on `256 (2**8)` eikä `64 (4**3)`.

11.3.2 Vertailuoperaattorit

Skalaarien ja muuttujien vertailu Perlissä voi olla puhtaasti numeerista tai vertailu voidaan suorittaa merkkijonotasolla (ASCII-vertailu). Kummallekin vertailutyypille on omat operaattorinsa. Perlissä tyhjä merkkijono (), nolla (`0`) ja "määrittelemätön" ovat *epätosia*, kaikki muut arvot ovat *tosia*.

Num.	**Mjono**	**Tulos**
`>`	`gt`	1, jos vasen puoli on *suurempi* kuin oikea
`>=`	`ge`	1, jos vasen puoli on *suurempi* tai *sama* kuin oikea
`<`	`lt`	1, jos vasen puoli on *pienempi* kuin oikea
`<=`	`le`	1, jos vasen puoli on *pienempi* tai *sama* kuin oikea
`==`	`eq`	1, jos vasen puoli on *sama* kuin oikea
`!=`	`ne`	1, jos vasen puoli *ei ole sama* kuin oikea
`<=>`	`cmp`	-1, jos vasen puoli on *pienempi*, 0, jos *sama* ja 1, jos *suurempi* kuin oikea

11.3.3 Merkkijono-operaattorit

Merkkijonoille Perl tarjoaa kaksi käyttökelpoista operaattoria: `.` ja `x`. Piste `.`-operaattorilla voidaan yhdistää kaksi merkkijonoa (katenointi). `x`-operaattori puolestaan toistaa merkkijonoa annetun lukumäärän mukaisesti. Esim.

```
#!/usr/local/bin/perl
$a="foo"; $b='bar';
print $a . $b . "\n"; # tulostaa foobar
print $a x 5 . "\n";  # tulostaa foofoofoofoofoo
```

11.3.4 Sijoitusoperaattorit

Kuten C-kielessä, myös Perlissä on lisäksi mahdollisuus käyttää ns. *sijoitusoperaattoreita*. Ts. mikä tahansa binäärinen operaattori voidaan yhdistää "yhtäsuuruusmerkkiin" (=), jolloin syntyy sijoitusoperaattori. Esim.

```
$a="foo"; $b='bar'; $c=3; $d=4;
$a .= $b;  # a:n arvo sijoituksen jälkeen "foobar"
$c -= $d;  # c:n arvo sijoituksen jälkeen -1
```

11.3.5 Muita operaattoreita

Seuraavassa on lueteltu vielä muutamia muita käyttökelpoisia operaattoreita.

Oper.	**Merkitys**
`++`	Lisäysoperaattori
`--`	Vähennysoperaattori
`!`	Negaatio (eri kuin)
`||`	Looginen OR
`&&`	Looginen AND
`?:`	`if-else`-ehto-operaattori
`//`	"regexp"-operaattori, merkkijonolauseke merkkien väliin.
`‘‘` tai `qx//`	Kuten shellissä, palauttaa merkkien välissä suoritettavien komentojen tulosteen.
`s///`	Korvaa-komento, toiminta kuten `sed`issä.
`y///` tai `tr///`	Muunna-komento, toiminta kuten `sed`issä.

11.4 Tiedosto-I/O

Kuten C-kielessä, tiedostojenkäsittely Perlissä perustuu *tiedostokahvoihin*. Perl määrittelee valmiiksi kolme perustiedostokahvaa:

- STDIN standardisyöte,
- STDOUT standarditulostus sekä
- STDERR standardivirhe.

11.4.1 `open` ja `close`-funktiot

Uuden tiedoston ja siis tiedostokahvan avaaminen tapahtuu `open`-funktion avulla. Tiedosto voidaan avata lukemista tai kirjoittamista varten. Lisäksi on mahdollisuus käyttää *putkituksia*.

`open(KAHVA, "tied_nimi")`	Tiedoston luku tiedostosta `tied_nimi`.
`open(KAHVA, "<tied_nimi")`	Sama kuin edellinen.
`open(KAHVA, ">tied_nimi")`	Kirjoitus tiedostoon. Vanha tuhoutuu.
`open(KAHVA, ">>tied_nimi")`	Kirjoitus tiedoston perään.
`open(KAHVA, "| ohjelma")`	Tulostuksen putkitus ulkoiselle ohjelmalle.
`open(KAHVA, "ohjelma |")`	Syötteen luku ulkoiselta ohjelmalta.

Tiedoston sulkemiseen on varattu funktio `close`. Funktio on muotoa `close(KAHVA)`, ja sitä on syytä käyttää varsinkin kirjoitettavaksi avattujen tiedostojen tapauksessa.

11.4.2 Tiedoston luku

Avattua tiedostoa voidaan lukea rivi kerrallaan `<>`-operaattorin avulla. Operaattorin syntaksi on muotoa:

```
<KAHVA>
```

Kahvan nimen käyttö on optionaalinen. Ilman kahvaa luetaan oletuksena komentoriviargumenttina annetusta tiedostosta (`@ARGV`-taulusta, jos määritelty) tai *stdin*stä. Rivin sisältö luetaan `$_`-muutujaan.

Kun kyseessä on koko tiedoston käsittely ja läpikäynti, suoritetaan lukeminen yleensä `while`-silmukassa (ks. Kontrollirakenteet). Esim.

```
while ($_ = <STDIN>) {
  # tehdään jotain riville esim.
  print $_;
}
```

11.4.3 Tiedostoon kirjoitus

Kirjoitettavaksi avattuun tiedostoon voidaan kirjoittaa `print` (ja `printf`) -funktion avulla. Tällöin `print`-funktiolle annetaan ensimmäiseksi parametriksi tiedoston `KAHVA`. Sitä seuraava osa on kirjoitettava osa. Esim.

```
open(OFILE, ">foobar"):
```

```
print OFILE "Tiedostoon kirjoitettava teksti";
```

11.5 Kontrollirakenteet

Seuraavassa on selvitetty Perlin tärkeimmät kontrollirakenteet. Merkintä `{ komennot }` tarkoittaa *blokkirakennetta* (blocks) ja voi siis sisältää useampia komentoja / lausekkeita:

```
{
  Lauseke1;
  Lauseke2;
  jne.
}
```

11.5.1 `if-elsif-else` ja `unless` -rakenteet

Perlin `if-else`-rakenne on seuraavanlainen.

```
if (ehtolause) {
  komennot
} elsif (ehtolause2) {
  komennot
} else {
  komennot
}
```

Sekä `elsif` että `else` ovat optionaalisia. Rakenne `unless (ehtolause) { komennot }` vastaa rakennetta `if (! ehtolause) { komennot }`.

11.5.2 `while`-silmukka

`while`-silmukan rakenne on tuttu:

```
while (ehtolause) {
  komennot
}
```

Rakenne `until (ehtolause) { komennot }` vastaa rakennetta `while (! ehtolause) { komennot }`. `while`-silmukka on samanlainen kuin C-kielessä.

11.5.3 `do-while`-silmukka

`do-while`-silmukka on rakenteeltaan seuraavanlainen.

```
do {
  komennot
} while (ehtolause);
```

Myös tässä tapauksessa rakenne `until (ehtolause) { komennot }` vastaa rakennetta `while (! ehtolause) { komennot }`.

11.5.4 `for`-silmukka

Myös `for`-silmukka on tuttu C-kielestä.

```
for (alustuslause; ehtolause; lisäyslause) {
  komennot
}
```

11.5.5 `foreach`-silmukka

`foreach`-silmukka on tarkoitettu listojen (taulukoiden) läpikäyntiin. Rakenteeltaan silmukka on seuraavanlainen.

```
foreach $alkio (@lista) {
  alkiolle suoritettavat komennot
}
```

Esimerkkinä taulukon tulostus.

```
#!/usr/local/bin/perl
@taulu=(a..z);
foreach $alkio (@taulu) {
  printf "$alkio\n";
}
```

11.5.6 Blokkien ja silmukoiden nimeäminen

Yksittäiset blokit sekä silmukat on mahdollista nimetä käyttäen erityistä *otsikkoa* blokin tai silmukan alussa. Nimeämisestä on se hyöty, että niihin pystytään viittaamaan muualta koodista käsin. Esim.

```
SWITCH: {
  ...
}
```

tai

```
RIVI: while (<STDIN>) {
  ...
}
```

11.5.7 Silmukan suorituksen ohjaus

Silmukoiden toimintaa voidaan ohjata komentojen `next`, `last` ja `redo` avulla. Komennot `next` ja `last` ovat lähellä C-kielen `continue` ja `break` -komentoja. Komento `next` tarkoittaa siis siirtymistä silmukassa seuraavalle kierrokselle, ilman että sen jälkeisiä komentoja suoritetaan. Komento `last` puolestaan siirtyy ulos silmukasta. Komento `redo` aloittaa silmukan komentojen alusta ilman, että uutta ehtolausetta arvioidaan. Komentojen syntaksi on seuraavanlainen.

```
next OTSIKKO ehtolause
last OTSIKKO ehtolause
redo OTSIKKO ehtolause
```

Sekä `OTSIKKO` että `ehtolause` ovat optionaalisia. Esimerkki `next`-funktion toiminnasta.

```
#!/usr/local/bin/perl
while (<STDIN>) {
  next if /^$/;   # ei käsitellä tyhjiä rivejä
  print           # tulostetaan muut rivit
}
```

Esimerkki "switch-case" rakenteen toteutuksesta.

```
SWITCH: {
  if (ehto_1) && do { komennot_1; last SWITCH; }
  if (ehto_2) && do { komennot_2; last SWITCH; }
  if (ehto_n) && do { komennot_n; last SWITCH; }
  oletus_komennot;
}
```

11.6 Funktiot

Perl sisältää suuren joukon sisäärakennettuja funktioita ja aliohjelmia. Käyttäjä voi myös luonnollisesti kirjoittaa omia aliohjelmia ja funktioita.

11.6.1 Omat aliohjelmat

Oma aliohjelma tai funktio on muotoa:

```
sub aliohjelman_nimi {
  komennot;
  return arvo;        # optionaalinen
}
```

Aliohjelmaa kutsutaan lisäämällä `&`-merkki funktion nimen eteen (optionaalinen versiossa 5). Aliohjelmalle voidaan välittää myös parametreja pilkulla erotettuna kaarisulkujen sisällä kuten C-kielessä. Parametrit saa käyttöön aliohjelmassa `@_`-taulukon avulla. Esim.

```
#!/usr/local/bin/perl

sub ynnaa {
  $sum=0;
  foreach $v (@_) {
    $sum+=$v;
  }
  return $sum;
}

printf "%d\n", ynnaa(1,2);
printf "%d\n", &ynnaa(2,3,4);
```

11.6.2 Sisäisiä funktioita

Seuraavassa muutamia hyödyllisistä Perlin sisäänrakennettuja funktioita.

`chop`	Poistaa merkkijonon viimeisen merkin. Käytetään usein poistamaan rivinvaihto-merkki “`\n`” merkkijonon, esim. rivin lopusta.
`die MSG`	Tulostaa ilmoituksen ja lopettaa suorituksen. Käytetään yleensä virhetilanteista ilmoitettaessa.

`each(AARR)`	Palauttaa (seuraavan) *indeksi-arvo*-parin (2-alkioinen taulukko) annetusta assosiatiivisestä taulukosta. Komennon avulla voidaan siis käydä assosiatiivisen taulukon alkiot läpi.
`exec(COM)`	Suorittaa annetun systeemikomennon, eikä koskaan palaa.
`exists $t{$i}`	Palautta TOSI, jos indeksi `$i` löytyy taulusta `%t`.
`exit(RET)`	Lopettaa suorituksen ja palaa arvolla `RET`.
`grep(re, @t)`	Palauttaa taulukon (tai täsmänneiden lkm) sisältäen taulukon `@t` ne alkiot, joihin merkkijonolauseke `re` täsmäsi.
`index(S,SS,P)`	Palauttaa ensimmäisen `SS`:n esiintymän paikan merkkijonossa `S` (tai paikan `P` jälkeen, jos käytössä, `P` on siis optio).
`join(FS,LIST)`	Splitin vastakohta. Liittää listan tai taulukon erillisiä merkkijonoja (`LIST`) yhdeksi merkkijonoksi, jossa kenttäerottimena `FS`.
`keys(AARR)`	Palauttaa taulukon, joka koostuu annetun assosiatiivisen taulukon indekseistä (nimistä).
`length(S)`	Palauttaa merkkien lukumäärän annetussa merkkijonossa. Jos parametria ei anneta, `$_`:n pituus.
`print FH LIST`	Tulostaa merkkijonon tai pilkulla erotellun listan merkkijonoja. `FH` on tiedostokahva,johon tulostetaan.
`printf FH LIST`	Kuten `print`, mutta mahdollisuus tulostaa formatoidusti, kuten C-kielessä.
`rand UL`	Palauttaa satunnaisluvun väliltä `0-UL`.
`reverse LIST`	Palauttaa parametrina annetun listan (esim. merkkijono) käänteisessä järjestyksessä.
`rindex(S,SS,P)`	Kuten `index`, mutta palauttaa viimeisen `SS`:n paikan merkkijonossa `S`.
`sleep SEC`	Laittaa skriptin nukkumaan parametrina sekunteina annetuksi ajaksi.
`sort LIST`	Palauttaa parametina annetun listan (esim. merkkijono) lajiteltuna.
`split(FS,STR)`	Jakaa merkkijonon taulukkoon. `FS` (voi olla regexp) ilmaisee kenttäerottimen, jonka mukaan merkkijono `STR` jaetaan.
`sprintf(FMT,LIST)`	Palauttaa `FMT`:n mukaan formatoidun merkkijonon listasta muuttujia (`LIST`).
`srand`	Asettaa siemenluvun `rand`-operaattorille.
`system(COM)`	Kuten `exec`, mutta ensin luodaan lapsiprosessi (fork). Jää odottamaan komennon paluuta.
`values(AARR)`	Palauttaa taulukon, joka koostuu annetun assosiatiivisen taulukon arvoista (vrt. `keys`).