Mathematica – Sekalaista asiaa

Asetusoperaattorit

Mathematicassa voi käyttää omia muuttujasymboleja melko rajattomasti ja niiden nimeämisessä voi käyttää miltei mitä tahansa merkkejä. Käytännössä nimeämisessä kannattaa pidättäytyä aakkosissa ja numeroissa, koska osa erikoismerkeistä on Mathematicassa varattu muihin tarkoituksiin. Esimerkiksi monissa ohjelmointikielissä ja komentotulkeissa käytettävää alaviivaa '_' ei voi käyttää muuttujasymbolin nimessä. Itsestään selvä rajoitus on tietysti myös se, ettei niminä kannata käyttää mitään, mikä on määritelty Mathematican laskentaytimessä sisäisesti tai jossain ulkoisessa paketissa. Toki minkä tahansa symbolinimen saa ottaa omaan käyttöönsä; tämä on mahdollista poistamalla halutun sisäisen symbolin suojaus. (`Unprotect[]`)

Arvon asettaminen muuttujasymbolille voidaan tehdä joko suoraan tai viivästetysti:

- Suora: `Symboli = Arvo`
- Viivästetty: `Symboli := Arvo`

Asetusoperaattorit poikkeavat toisistaan olennaisesti. Suorassa asetuksessa symbolille määrätään arvo asetushetkellä ja arvo pysyy kiinteänä tämän jälkeen. Mikäli symbolin arvo riippuu jostain toisesta symbolista, ei tämän toisen symbolin arvon muuttaminen vaikuta kiinteästi (suoraan) asetettuun arvoon. Viivästetyssä asetuksessa sen sijaan symbolin arvo lasketaan uudestaan joka kerta kun symboliin viitataan. Esimerkiksi:

```
x = 2; luku1 = x^2 ; luku2 := x^2;
{luku1, luku2} ⟶ { 4, 4 }
x = 3; {luku1, luku2} ⟶ { 4, 9 }
```

Viivästetyllä asetuksella asetettu luku siis muuttui kun x:n arvoa muutettiin. Viivästettyä sijoitusta tulee käyttää erityisesti omia funktioita määriteltäessä, sillä funktion arvohan on tarkoitus laskea jokaisella viittauskerralla uudestaan.

Symbolien poistaminen muistista

Kun symboleja otetaan käyttöön, on hyvä miettiä kuinka kauan symbolia tarvitsee. Muistiin unohtuneet symbolit sekoittavat helposti laskutoimituksia ja hidastavat työskentelyä myöhemmin. Symbolit tulisi siis heti käytön jälkeen poistaa muistista `Clear[]`-komennolla.

Miten sitten voi muistaa kaikki asettamansa symbolit, ellei niitä vapauta heti? Yksi vaihtoehto on ylläpitää asettamistaan symboleista listaa, johon jokainen

uusi symboli lisätään ja josta vanhat poistetaan. Samoin tämän listan läpikäynnillä on helppo vapauttaa kaikki muuttujat yhtaikaa. Saman menettelyn voi toki hoitaa muutenkin, mutta yksi tapa toteuttaa mainitunlainen lista olisi tehdä seuraavat määritykset:

```
S={};
NS[sym_,value_]:=Do[AppendTo[S,ToString[sym]]; sym=value;];
Clr:=Do[Apply[ClearAll,S];S={}];
```

Kun nämä on tehty, voidaan uusia symboleja ottaa käyttöön komennolla `NS[]` (suora asetus). Kaikki näin määritellyt symbolit voidaan poistaa muistista komennolla `Clr`.

```
NS[x, 2]; NS[luku1, x^2 ]; NS[luku2, x^2];
{luku1, luku2} ⟶ { 4, 4 }
x = 3; {luku1, luku2} ⟶ { 4, 9 }
Clr
```

Esitetty on vain yksi mahdollisuus pitää kirjaa määrittelemistään symboleista. Mathematicalla on mahdollista tehdä asia myös muilla tavoilla. Tärkeintä on ymmärtää se, että symbolit on syytä poistaa muistista heti kun niitä ei enää tarvitse.

Muunnos ja Korvausoperaattorit

Mathematican sisäiseen logiikkaan kuuluvat myös muunnokset (`Rule`). Käytössä voi tavata ulkoasultaan seuraavan näköisiä olioita:

```
Muunnettava → Muuntotulos
```

Esimerkki tällaisesta on yhtälön ratkaisu:

```
tulos = Solve[x^3 - xy==0, y] ⟶ {{y → x^2}}
```

Tuloksena ei siis ole selkeä lauseke vaan *muunnos*. Muunnoksen voi tulkita lausekkeeksi *korvausoperaattorilla*. Näitä on kahta tyyppiä:

- Tavallinen: `Symboli /. Muunnos` (`ReplaceAll`)
- Rekursiivinen: `Symboli //. Muunnos` (`ReplaceRepeated`)

Tavallinen korvaus muokkaa alkuperäistä symbolia muunnoksen antaman ohjeen mukaisesti pysähtyen yhden korvauksen jälkeen. Rekursiivinen korvaus puolestaan jatkaa niin pitkään kuin korvattavaa on. Rekursiivisesti laskettaessa on aina kuitenkin varmistettava, että rekursiokierto loppuu jossain vaiheessa. Ellei

rekursiota jossain vaiheessa katkaista, joudutaan ikuiseen silmukkaan. Havainnollistetaan muunnoksen käsitettä laskemalla muunnoksen avulla esimerkiksi Gaussin summa $1 + 2 + \ldots + 100$:

```
muunnos = f[x_] → x + f[x-1];
f[100] /. muunnos ⟶ 100 + f[99]
f[0] = 0;
f[100] //. muunnos ⟶ 5050
Clear[muunnos], ClearAll[f]
```

Myös eri komennoille annettavat optiot ovat muunnoksia. Jotkut komennot (esim. edellä nähty `Solve`) taas palauttavat arvoinaan muunnoksia. Esimerkiksi edellä saatu ratkaisu

```
tulos = Solve[x^3 - x y ==  0, y] ⟶ {{y → x^2}}
```

on lista muunnoksia. Yhtälöllä voisi olla useampia ratkaisuja, mutta tässä tapauksessa ratkaisulistassa on vain yksi muunnos. Saatu tulos voidaan piirtää koordinaatistoon käyttämällä korvausoperaattoria:

```
Plot[y /. tulos, {x,-2,2}]
```

Muuttujien tyypitys l. Muuttujamallit (`Pattern`)

Aiemmin on jo käytetty alaviivoja '_' muuttujasymbolien nimien yhteydessä. Nämä tarkoittavat muuttujamalleja, joilla rajataan käytettävän symbolin datatyyppiä. Yksinkertaisin malli on jo aiemminkin käytetty tyhjä alaviiva. Joissakin tapauksissa on kuitenkin hyödyllistä rajata operaatiot koskemaan vain tietyn tyyppisiä symbolien arvoja. Esimerkiksi aiemmin tehty rekursio

```
muunnos = f[x_] → x + f[x-1];
f[0] = 0;
```

on tarkoitettu kokonaisluvuille. Jos rekursiota kutsutaan jollain muulla kuin kokonaislukuarvolla, joudutaan virhetilanteeseen. Esimerkiksi komento

```
f[100.5] //. muunnos
```

johtaa virhetilanteeseen, koska rekursion katkaisukohta on määrätty paikkaan, jossa parametrin arvo on nolla (kokonaisluku). Arvolla 100.5 aloitettaessa rekursio jatkaa ykkösen välein alaspäin ja hyppää nollan yli:

```
... ⟶ 2.5 + f[1.5]
f[1.5] /. muunnos ⟶ 1.5 + f[0.5]
f[0.5] /. muunnos ⟶ 0.5 + f[-0.5]
f[-0.5] /. muunnos ⟶ -0.5 + f[-1.5]
f[-1.5] /. muunnos ...
```

Laskenta ei siis pysähdy koskaan. Jos sen sijaan kiinnitetään muuttujatyyppi positiiviseksi kokonaisluvuksi, ei virheitä pääse syntymään:

```
muunnos = f[x_Integer /; x>0] → x + f[x-1];
f[0] = 0;
Clear[muunnos]
```

Nyt virheelliset syötteet eivät suorita korvausta:

```
f[100] ⟶ 5050
f[100.0] ⟶ f[100.]
f[-5] ⟶ f[-5]
```

Tyyppejä voidaan rajoittaa laajemminkin käyttäen mitä tahansa Mathematican tuntemia tietorakenteita tai komentoja. Esimerkiksi

```
f[lista_List, n_Integer] := lista^n
f[{1,2,3}, 3] ⟶ {1,8,27}
f[{1,2,3}, 2.5] ⟶ f[{1,2,3}, 2.5]
f[3, 2.5] ⟶ f[3, 2.5]
Clear[f]
```

Rajoittamiseen voi käyttää myös erilaisia valmiita testifunktioita. Esimerkiksi `EvenQ` testaa, onko annettu parametri parillinen. Tällaisen testifunktioiden avulla voidaan syötteitä rajata hyvinkin tarkasti.

```
f[x_Integer ? EvenQ] := x/2
f[6] ⟶ 3
f[7] ⟶ f[7]
Clear[f]
```

Funktio voidaan määritellä myös paloittain:

```
f[x_ /; x <= 0] := x
f[x_ /; x > 0] :=x^2
Clear[f]
```

Rekursiiviset funktiot

Rekursiota voi Mathematicassa tehdä myös perinteiseen tapaan määrittelemällä funktioita, jotka kutsuvat itseään. Myös tällöin on huolehdittava rekursion katkeamisesta ja mahdollisesta muuttujatyypityksestä mallien avulla.

Edellä laskettu Gaussin summa voidaan siis laskea myös seuraavasti:

```
Summa[n_Integer /; n > 0] := n + Summa[n-1]
Summa[0] = 0;
Clear[Summa];
```

Komentojen optioista

Mathematican komentojen toimintaan voi yleensä vaikuttaa optioilla. Komennon optioita voi tiedustella komennolla `Options[komento]`. Tämä tuo esiin optiolistan, josta käytettävissä olevat optiot oletusarvoineen näkyvät. Optioiden toimintaan ja käyttöön löytyy apua help-järjestelmästä. Yleisin paikka käyttää optioita on grafiikkaa tuottavat komennot. Esimerkiksi alla olevassa tilanteessa joudutaan käyttämään optioita, että saadaan näkyviin pelkästään haluttu kuva:

```
pisteet=Table[Random[],{6}];
paraabeli=Fit[pisteet,{1,x,x^2},x];
kuva1=ListPlot[pisteet,DisplayFunction->Identity];
kuva2=Plot[paraabeli,{x,0,7},DisplayFunction->Identity];
Show[kuva1,kuva2,DisplayFunction->$DisplayFunction,
      Prolog->AbsolutePointSize[6]];
Clear[pisteet,paraabeli,kuva1,kuva2]
```

Print ja Input

Jos tarvitaan interaktiivisia toimintoja, voidaan Mathematicalla kysyä tietoja käyttäjältä. Samoin voidaan tulostaa halutun kaltaisia tulosteita. Näitä varten on esim. komennot `Print` ja `Input`. Alla yksinkertainen esimerkki näiden käytöstä:

```
Do[n=Input[“Anna kokonaisluku: ”];
    Print[“Luvun ”, n, “ kertoma on ”, n!, “.”];
```

Map ja Apply

Jos halutaan tehdä joku tietty operaatio suurelle määrälle alkioita, voidaan nämä alkiot pakata listaksi ja sen jälkeen käyttää `Map`-komentoa. Vastaavasti,

jos halutaan suorittaa jokin komento käyttäen komennon parametrina jotain listaa, voidaan käyttää `Apply`-komentoa.

```
f[x_]:=x^2;
Map[f, Range[100]] ⟶ {1,4,...,10000}
Apply[Plus, Range[100]] ⟶ 5050
Clear[f]
```

Module ja Block

Mathematican komentoja voidaan paketoida aliohjelman kaltaisiksi ohjelmamoduleiksi, joita voidaan käyttää aivan kuten ohjelmointikielten aliohjelmia. Tällaisia rakenteita varten on olemassa komennot `Module` ja `Block`. Näille luetellaan käytettävät muuttujat sekä ajettavat komentorivit.

```
f[l_List,n_Integer]:=Module[{x},x:=Take[l,-n]; Reverse[x]];
f[{a,b,c,d,e},3] ⟶ {e,d,c}
Clear[f]
```

Muuttujat voidaan myös alustaa muuttujalistaa lueteltaessa. Mathematican symbolit (muuttujat) ovat yleisesti globaaleja, mutta ohjelmalohkon sisäiset muuttujat ovat lokaaleja. Ne eivät siis jää laskentaytimen muistiin ohjelmalohkon suorituksen jälkeen ohjelmalohkon ulkopuolelle. `Module` ja `Block` toimivat eri tavoin lokaalien muuttujien alustuksessa. Mikäli lokaalina käytetty muuttuja on jo käytössä ohjelmalohkon ulkopuolella, alustaa `Block` myös sisäiset muuttujansa näillä globaaleilla arvoilla. `Module` puolestaan pitää muuttujansa tässäkin suhteessa lokaaleina.

```
x=2;
Module[{x,i=3}, i+x] ⟶ 3 + x$1
Block[{x,i=3}, i+x] ⟶ 5
Clear[x]
```

Do, While, For, If, ...

Mathematicassa voi myös käyttää ohjelmointikielten silmukka- ja ehtorakenteita. Ohjelmalohkoista voi rakentaa erittäin monipuolisia. Näistä esimerkkejä help-järjestelmässä ja harjoituksissa...