Eräs APL-opas

Johdanto

Tämän oppaan tarkoituksena on tarjota tiivis tietopaketti APL-kielestä, sen erityispiirteistä, ominaisuuksista ja säännöistä. APL-kirjallisuutta on kautta aikain suomennettu vähän, mutta sitäkin paremmin: Gustav *Tolletin* suomentama "se vihreä" APL-kielen opas on edelleen täyttä rautaa. I-APL -käsikirja lienee viimeisin varsinainen käännös, useita luentomonisteita on tehty eri tarpeisiin. Ulkomaista APL-kirjallisuutta on saatavilla sitten sitäkin enemmän.

Tätä **ei** ole tarkoitettu kaikenkattavaksi APL-oppikirjaksi – sellaisen tekemiseen ei ole resursseja – itseopiskelun ja kertauksen tueksi kylläkin. Olen koonnut yhteen arvokkaina pitämiäni tietohippuja eri lähteistä: käsikirjoista, kirjallisuudesta, konferenssijulkaisuista, seminaaripapereista ja lehdistä. Luettelo tärkeimmistä lähdeteksteistä ja muusta suositeltavaksi katsomastani materiaalista on tämän oppaan lopussa.

Lappeenrannan Teknillisen Korkeakoulun APL-kurssin 15-sivuinen luentopruju, Seppo *Linnainmaan* tekemä, on se opas, josta olen perustietoni ammentanut. Aloin aikoinani siirtää tätä käsinkirjoitettua pikku tiiliskiveä elektroniseksi; kun olin vihdoin saanut pääosat naputelluksi, kasvoi ruokahalu niin että aloin rakentaa samaan muottiin laajempaa kokonaisuutta.

Tämmöinen siitä sitten tuli.

Opas koostuu kahdesta pääosasta: APL (standardi-APL) ja APL2. Käsittelytapaan on vaikuttanut vahvasti APL:n kehityskaari, eli ns. historiallisesti pakottavat syyt.

Alkuosassa tutustutaan kielen periaatteisiin ja porhalletaan lyhyesti perusfunktiot läpi. Skalaarifunktiot, joihin esimerkiksi jo kansakoulusta tutut aritmeettiset funktiot kuuluvat, esitetään taulukkoina ja sekafunktiot käsitellään ensin vektoriargumenttien (tietojonojen) kanssa. Sitten tutustutaan moniulotteisiin sääntiöihin ja käydään sekafunktiot uudestaan tarkemmin ja muodollisemmin läpi. Itse ohjelmoitavan funktion rakenne, operaattorit, järjestelmämuuttujat ja -funktiot sekä järjestelmäkomennot esitellään pintapuolisesti. Lopussa on vielä pieni idiomilista, hyödylliseksi havaittuja lyhyitä APL-ilmaisuja.

APL2-osassa perehdytään IBM:n de facto -standardin saavuttaneeseen toisen sukupolven APL-kieleen.

Teoriaa on joltisensakin paljon: APL2:n kehitysperiaatteet ja uudet käsitteet (mm. sisäkkäisyys) esitellään tiiviisti. Olen tähän yhteyteen tuonut sellaisia tekstejä, joita ei tietääkseni ole suomeksi ennen laajasti esitetty, siitä tuo teoreettisuuden ylenmääräinen kalkinpöly. Lopuksi esitellään uudet funktiot, operaattorit, järjestelmäfunktiot ja -muuttujat ynnä systeemikäskyt sekä muutama APL2-idiomi.

Parhaan hyödyn tästä oppaasta saa, jos voi samanaikaisesti harjoitella jollain APL-tulkilla. Eri valmistajien tuotteilla ovat omat (käsikirjoissa kerrotut) rajoituksensa ja kielilaajennuksensa, joten jotkut esimerkit eivät välttämättä toimi juuri prikulleen kuten tässä oppaassa. Pääperiaatteet kuitenkin ovat samat.

Opas on kirjoitettu hyvässä uskossa sellaisen tarpeeseen, sen painotukset, kieliasu ja mahdolliset virheet ovat omiani. Arvokasta oikoluku- ja huomautusapua ovat Jaanaliisan (vaimon) lisäksi antaneet *Juvosen* Arto, *Pasasen* Pauli, *Pakarisen* Perttu ja *Linnan* Kimmo. Kiitoksien arvoisesti.

Pitäkää hyvänänne.

Järvenpäässä 19.10.1998

Veli-Matti Jantunen

3.20/5.8.02

1. APL

Yleistä

A Programming **L**anguage (Kenneth E. *Iverson*: 1957 –).

Alun perin matemaattinen **notaatio** (merkintätapa), ei tavanomaisessa mielessä ohjelmointikieli. APL:n johtavana periaatteena on ilmaista matemaattiset ja tietojenkäsittelylliset perustoiminnot omalla funktiosymbolillaan tai niiden yhdistelmillä; luku- ja merkkijoukkoja operoidaan **kokonaisuuksina** (siitä kutsumanimi *Array Processing Language*).

APL:ää voidaan käyttää:

- välittömään laskentaan (mahdollistaa tietokoneen laskimenomaisen käytön)
- itse tehtyjen **funktioiden** (ohjelmien) määrittelyyn ja ajoon.

APL-funktiot voivat olla:

- **monadisia** eli yksiargumenttisia (esim. `*X` tarkoittaa e^x:ää)
- **dyadisia** eli kaksiargumenttisia (esim. `X*Y` tarkoittaa x^y:tä)
- **niladisia** eli argumentittomia.

Useimmat funktiosymbolit ovat käytössä sekä monadisina että dyadisina, yksikään perusfunktio (primäärifunktio) ei ole argumentiton. Funktiot ovat keskenään **samanarvoisia**, niillä ei ole erillistä funktiohierarkiaa.

APL-lausekkeiden suoritusjärjestys on aina **oikealta vasempaan**, ellei sulkein toisin osoiteta; toisin sanoen funktion oikeana argumenttina on sen oikealla puolella olevan lausekkeen arvo. Tarvittaessa voidaan käyttää ylimääräisiä sulkeita lausekkeiden suoritusjärjestyksen selventämiseksi.

APL toimii tulkkiperiaatteella: ohjelmia ei käännetä vaan ne suoritetaan alkuperäiskoodia suoraan tulkitsemalla. Tämä mahdollistaa ohjelmakoodin ja muuttujien tarkastelun tai muokkauksen **kesken suorituksen**. Sovellus voidaan keskeyttää (tahallaan tahi virhetilanteeseen) ja suoritusta voidaan jatkaa tehtyjen korjausten jälkeen suoraan keskeytyskohdasta. APL on ollutkin aina hyvin tehokas **prototyyppikehitin**.

APL-vakiot

Numeeriset skalaarit:

- nollaa pienemmillä luvuilla on etumerkkinä **negatiivi** (ylämiinus) `¯`
- reaaliluvun desimaalierotin on piste (`3.14159`)
- skaalattujen lukujen kokonaislukueksponenttiosa erotetaan *E*:llä (`7.5E¯3 = 0.0075`)
- sama luku voidaan syöttää usealla eri esitystavalla (`17.0 = 1.7E1 = 17`)
- luvun osat kirjoitetaan yhteen (ei välilyöntejä, pilkkuja tms.).

Numeeriset vektorit (lukujonot):

- joukko **välilyönnein** erotettuja lukuskalaareja (`3 3.141 7.5E¯3`).

Tekstiskalaari:

- heittomerkkien välissä oleva yksittäinen merkki (`'A'`).

Tekstivektorit:

- heittomerkkien ympäröimä merkkijono (`'TEKSTIÄ'`)
- merkkijonon sisältämät heittomerkit kirjoitetaan kahdennettuina (`'TIU''UT'`).

Totuusarvot:

- totuusarvoja vastaavat (loogiset) binääriluvut `1` (tosi) ja `0` (epätosi)
- voidaan käyttää sekä loogisesti (`1∨0`) että aritmeettisesti (`1+1`).

Dyadiset aritmeettiset skalaarifunktiot

merkki	*nimitys*	*määritelmä*	*esimerkki*	*tulos*
+	yhteenlasku *(add)*	x+y	2+3	5
-	vähennyslasku *(subtract)*	x-y	2-3	¯1
×	kertolasku *(multiply)*	x·y	2×3	6
÷	jakolasku *(divide)*	x/y	2÷¯5	¯0.4
|	jakojäännös *(residue)*	y mod x	1|2.17	0.17
⌈	maksimi *(maximum)*	max(x,y)	3⌈7	7
⌊	minimi *(minimum)*	min(x,y)	¯2⌊¯1	¯2
⋆	potenssi *(power)*	x^y	9⋆0.5	3
⍟	logaritmi *(logarithm)*	$\log_x y$	10⍟2	0.30103..
!	binomikerroin *(binomial)*	x!/(y!·(x-y)!)	2!5	10

HUOM: 0÷0 = 1 0|*X* = *X* 0⋆0 = 1

Monadiset aritmeettiset skalaarifunktiot

merkki	*nimitys*	*määritelmä*	*esimerkki*	*tulos*
+	arvo (luku) *(value, identity)*	0+x	+3	3
-	vasta-arvo (vastaluku) *(negate)*	0-x	-¯2	2
×	etumerkki *(signum)*		×¯3	¯1
÷	käänteisarvo (käänteisluku) *(reciprocal)*	1/x	÷2	0.5
|	itseisarvo *(absolute value, magnitude)*	$\lvert x \rvert$	|¯2.1	2.1
⌈	yläpyöristys (yläarvo, katto) *(ceiling)*	$\lceil x \rceil$	⌈¯7.7	¯7
⌊	alapyöristys (ala-arvo, lattia) *(floor)*	$\lfloor x \rfloor$	⌊11.9	11
⋆	eksponenttifunktio *(exponential)*	e^x	⋆1	2.71828..
⍟	luonnollinen logaritmi *(natural logarithm)*	ln x	⍟⋆5	5
!	kertoma (gammafunktio x+1) *(factorial)*	x!	!3	6
?	satunnaisluku (joukosta [1..x]) *(roll)*		?6	2 (esim.)

Dyadiset loogiset skalaarifunktiot, relaatiot

merkki	*nimitys*	1 α 1	1 α 0	0 α 1	0 α 0
∧	ja (looginen tulo) *(and)*	1	0	0	0
∨	tai (looginen summa) *(or)*	1	1	1	0
⍲	poissulkeva ja (ei-ja) *(nand)*	0	1	1	1
⍱	poissulkeva tai (ei-tai) *(nor)*	0	0	0	1
<	pienempi *(less than)*	0	0	1	0
≤	pienempi tai yhtä suuri (ei suurempi, looginen implikaatio ⟹) *(less than or equal)*	1	0	1	1
=	yhtäsuuri (yhtä kuin) *(equal)*	1	0	0	1
≥	suurempi tai yhtä suuri (ei pienempi) *(greater than or equal)*	1	1	0	1
>	suurempi *(greater than)*	0	1	0	0
≠	eri suuri *(not equal, xor)*	0	1	1	0

Monadiset loogiset skalaarifunktiot

merkki	*nimitys*	*määritelmä*	*esimerkki*	*tulos*
~	negaatio (ei) *(not)*	¬x	`~0`	`1`

HUOM: `1` = tosi *(true)*, `0` = epätosi *(false)*

Trigonometriset funktiot (pallo-, ympyräfunktiot) ○

merkintä	*määritelmä*	*merkintä*	*määritelmä*
`○X`	π × x (pii kertaa) *(pi times)*		
`0○X`	$\sqrt{1-x^2}$		
`1○X`	sin x	`¯1○X`	arcsin x
`2○X`	cos x	`¯2○X`	arccos x
`3○X`	tan x	`¯3○X`	arctan x
`4○X`	$\sqrt{1+x^2}$	`¯4○X`	$\sqrt{-1+x^2}$
`5○X`	sinh x	`¯5○X`	arsinh x
`6○X`	cosh x	`¯6○X`	arcosh x
`7○X`	tanh x	`¯7○X`	artanh x

HUOM: Kulman arvo esitetään **radiaaneina**, esimerkiksi sin 30º: `1○○30÷180 = 0.5`

Sijoitus (asetus, olkoon) ←

Sijoitusfunktio *(assignment)* on dyadinen funktio, jonka vasen argumentti on sen muuttujan nimi johon oikean argumentin (lausekkeen) arvo sijoitetaan.

Muuttujan nimen (kuten funktionkin) ensimmäisenä kirjaimena on jokin seuraavista merkeistä:
`A..Z, a..z, ∆, ⍙` ja seuraavina merkkeinä voi näiden lisäksi olla joku seuraavista: `0..9,_`.
Pienten kirjainten sijaan on alun perin käytetty *A̲L̲L̲E̲V̲I̲I̲V̲A̲T̲T̲U̲J̲A̲* kirjaimia, jotka vieläkin ovat monissa APL-tulkeissa käytössä. Muuttujan nimen pituudella ei ole yleensä käytännössä esiin tulevia rajoituksia.

Nimissä **ei** käytetä välilyöntejä.

Muuttujan tietotyyppiä tai kokoa **ei määritellä etukäteen**, muuttujan arvo voi olla numeerinen, looginen tai merkkimuotoinen skalaari tai vektori tahi usean skalaarin muodostama taulukko yleisnimitykseltään **sääntiö**.

Sijoitusfunktion arvo = oikean argumentin arvo. Sijoitus voi esiintyä missä tahansa lausekkeen sisällä.
Jos sijoitus on **viimeinen** suoritettava (vasemmanpuoleisin) funktio, **ei** lausekkeen arvoa tulosteta ruudulle.

```
      B←2+C←9-A←5
      A+B+C
15
      ISOONTALON_ANTTI_JA_RANNANJARVI←'RAA''AT HELAPÄÄLEU''UT'
      ISOONTALON_ANTTI_JA_RANNANJARVI
RAA'AT HELAPÄÄLEU'UT
```

HUOM: Syöte esitetään perinteisesti kuuden merkin verran sisennettynä ja tulos vasempaan reunaan tasattuna. Tulostettavaa merkkidataa **ei** rajata heittomerkeillä.
Uudet APL-tulkit **saattavat** sallia kansallisten erikoismerkkien käytön nimissä (`YÖ←1`).

Luku- ja kirjoituspyyntö ⎕ ⍞

Merkintä ⎕←*X* ilmaisee, että *X*:n arvo halutaan **tulostaa** ruudulle *(evaluated output, quad).*

```
      ⎕←B←5-3
2
```

Kirjoituspyyntöä ⎕ ("luukku/ovi/ikkuna/näyttö") ei aina tarvita: jos APL-lauseke ei päädy sijoitusfunktioon, sen arvo tulostetaan ruudulle. Välitulosten esittämiseen ja selkeyden takia sitä kuitenkin kannattaa käyttää.

```
      ⎕←2×⎕←2*10
1024
2048
```

Jos kirjoituspyyntösymbolin oikealla puolella ei ole sijoitusnuolta, sen arvo on näppäimistöltä **syötettävän lausekkeen** arvo. Lukupyyntönä *(evaluated input, quad)* APL-istuntoon tulostuu tällöin ⎕: .

```
      A←3
      ⎕←2+⎕
⎕:  A+5
10
```

Jos rivinpalautusta ei haluta suorittaa tulostuksen jälkeen, se saadaan estetyksi käyttämällä ⎕:n sijaan merkki-näyttösymbolia ⍞ *(character output, quote quad).* Jos ⍞:n oikealla puolella ei ole sijoitusnuolta, sen arvo on näppäimistöltä luettavan tekstin sisältö **merkkijonona** *(character input, prompt, quote quad).*
Merkintää ⎕: **ei** tällöin tulostu.

```
      ⎕←B←⍞
A+5
A+5
```

Vektoriargumenttiset skalaarifunktiot

Jos monadisen skalaarifunktion argumentti on vektori, kohdistuu funktio kuhunkin sen alkioon **erikseen**.

```
      V←1 2 3 4 5
      !V
1 2 6 24 120
```

Jos dyadisen skalaarifunktion molemmat argumentit ovat vektoreita, niissä pitää olla **yhtä monta** alkiota. Skalaarifunktion tuloksena on vektori, jossa kukin alkio on saatu soveltamalla kyseistä funktiota argumenttien **vastinalkioihin**.

Jos toinen argumenteista on skalaari tai yksialkioinen vektori, se käsitetään laajennetuksi toisen argumentin kokoiseksi vektoriksi ennen funktion suoritusta (ns. **skalaarilaajennus**).

```
      V×2
2 4 6 8 10
      'ABRAKADABRA'='A'
1 0 0 1 0 1 0 1 0 0 1
      A←3 1 0 1 9 5
      B←1 8 1 2 9 6
      A×B
3 8 0 2 81 30
```

Vektoriargumenttiset sekafunktiot

indeksointi (viittaus vektorin alkioihin) `[ ]`

Viitattavan alkion järjestysnumero kirjoitetaan vektorin perään **hakasulkeisiin**.

```
            X←17 13 15 11 18
            X[2]
13
```

Jos hakasulkeissa on järjestysnumero**vektori**, tuloksena on vastaavista alkioista muodostettu tulos**vektori**.

```
            X[4 2 3 4]
11 13 15 11
            A←'KARHU'
            ⎕←T←A[4 2 2 3 5 1 1 2]
HAARUKKA
```

Olemassa olevan vektorin yksittäisten alkioiden arvoja voidaan muuttaa hakasuljeindeksoinnin ja sijoitusnuolen avulla.

```
            A←'KAHVI'
            A[4 1]←'TS'
            A
SAHTI
            T[1 3 2 5]←'PIIA'
            T
PIIRAKKA
```

koko, koonti `ρ`

Monadinen kokofunktio `ρ` (”rho”) kertoo vektorin alkioiden lukumäärän.

```
            ρ1 2 3 4 5 6 7 8 9 10
10
            T[ρT]
A
```

Dyadinen koontifunktio `NρX` **toistaa** `X`:n sisältöä, kunnes tulosvektorissa on positiivisen kokonaisluvun `N` verran alkioita.

```
            4ρ'A'
AAAA
            6ρ'KAS'
KASKAS
            3ρ1 2 3 4 5
1 2 3
```

Vektori, jossa ei ole yhtään alkiota, on **tyhjä vektori**. Tyhjän tekstivektorin saa kirjoittamalla kaksi rajoitinheittomerkkiä perätysten: `''` (välissä ei saa olla mitään). Tyhjän numeerisen vektorin saa vaikkapa koontifunktiolla (istuntoon tulostuu tässä tyhjä rivi):

```
            0ρ1
```

otto ↑

Ottofunktiolla N↑X otetaan |N alkiota vektorin X alku- (N>0) tai loppupäästä (N<0).
Jos (|N)>ρX, täydennetään tulosta X:n tyypin mukaan joko nollilla tahi välilyönnein.

```
            5↑'VIRTAPIIRI'
VIRTA
            ¯5↑'HAUPITSI'
PITSI
            A←2 4 6 8 10
            7↑A
2 4 6 8 10 0 0
            ¯1↑A
10
            ¯7↑A
0 0 2 4 6 8 10
```

pudotus ↓

Pudotusfunktiolla N↓X poistetaan |N alkiota vektorin X alku- (N>0) tai loppupäästä (N<0).

```
            3↓'VIRTAPIIRI'
TAPIIRI
            ¯2↓A
2 4 6
            3↓7↑A
8 10 0 0
```

Jos (|N)≥ρX, on tuloksena aina **tyhjä** vektori.

```
            ρ123↓A
0
```

jäsenyys (joukkoon kuuluminen) ∊

X∊V kertoo, onko skalaari X yhtä suuri kuin jokin vektorin V alkioista (tulos 1) vai ei (tulos 0).

Jos X on vektori, vertailu suoritetaan **jokaiselle** X:n alkiolle erikseen ja tuloksena on X:n pituinen looginen (bitti-)vektori.

```
            A←'VESIHIISI SE SIHISI HISSISSÄ'
            B←'AEIOUYÄÖ'
            'Ä'∊A
1
            'Å'∊B
0
            A∊B
0 1 0 1 0 1 1 0 1 0 0 1 0 0 1 0 1 0 1 0 0 1 0 0 1 0 0 1
            B∊A
0 1 1 0 0 0 1 0
```

liitos ,

Lauseke *X*,*Y* antaa tulokseksi vektorin, jossa *Y*:n alkiot on liitetty *X*:n alkioiden perään.

```
        A←3 4 5
        B←101 102
        A,0,B
3 4 5 0 101 102
        C←'KOKO'
        C,'A ',C,'ON ',C,' RO',C,'O',C,'US!'
KOKOA KOKOON KOKO ROKOKOOKOKOUS!
```

lukusarja, sijainti ⍳

Monadinen lukusarjafunktio ⍳*X* ("iota") tuottaa **positiivisesta** kokonaisluvusta *X* vektorin, jonka alkioina ovat luvut 1..*X*.

```
        ⍳9
1 2 3 4 5 6 7 8 9
        20+(⍳7)÷10
20.1 20.2 20.3 20.4 20.5 20.6 20.7
```

Jos *X* on nolla, on tuloksena **tyhjä** vektori (⍳0) jonka koko = 0.

Dyadisen sijaintifunktion *V*⍳*X* tuloksena on *X*:n kanssa samanmuotoinen sääntiö, jonka kukin alkio ilmaisee monentenako vastaava *X*:n alkio on vektorissa *V* (useista esiintymistä **ensimmäinen**).

Jollei alkiota löydy ollenkaan, on tuloksena **yhtä suurempi** luku kuin *V*:ssä on alkioita.

```
        'ACDC'⍳'C'
2
        2 3 8 9⍳⍳4
5 1 2 5
        ';,.;:'⍳'. ;'
3 6 1
```

satunnaisluku ?

Monadinen satunnaislukufunktio ?*X* tuottaa **positiiviselle** kokonaisluvulle *X* satunnaisluvun väliltä [1,*X*].

Esimerkiksi viiden nopan samanaikaista heittoa voi simuloida seuraavasti:

```
        ?5⍴6
3 3 3 3 3
```

Yatzy!

supistus /

Kaksiargumenttinen supistusfunktio *V/X* valitsee *X*:stä loogisen ohjainvektorin *V* ykkösiä vastaavat alkiot. Argumenteissa on oltava yhtä monta alkiota tai sitten *V* voi olla skalaari, jolloin koko oikea argumenttivektori supistetaan tällä arvolla.

```
      A←5 4 3 2 1
      1 0 1 0 1/A
5 3 1
      C←'MAAPALLO'
      0 0 1 1 0 0 1 0/C
APL
```

pelkistys (reduktio) α/

Jos vektorilla halutaan suorittaa laskutoimitus, joka voitaisiin suorittaa panemalla joka alkioväliin sama **skalaarifunktio** α, tämä funktio voidaan panna kenoviivalla erotettuna vektorin eteen.

```
      +/A
15
      ×/A
120
```

Funktioargumenttinen pelkistys on itse asiassa **operaattori**. Niihin palataan myöhemmin.

lavennus \

Kaksiargumenttisen lavennusfunktion *V\X* loogisessa ohjainvektorissa *V* on oltava yhtä monta ykköstä kuin *X*:ssä on alkioita (**ei** skalaarilaajennusta). Lavennuksen tuloksena on vektori, jossa *V*:n nollaa vastaaviin kohtiin tulee *X*:n tyypin mukaan joko nollia tai välilyöntejä.

```
      1 0 1 0 1\'ABC'
A B C
      1 1 0 0 1 0 1 1 0\A
5 4 0 0 3 0 2 1 0
```

selaus (kertymä) α\

Jos lavennuksen vasen argumentti korvataan skalaarifunktiomerkinnällä (α\X), on tuloksena *X*:n kokoinen kertymävektori, jossa kukin alkio edustaa osavektorien (X[1], X[1 2], X[1 2 3], ..) pelkistystä kyseisellä skalaarifunktiolla.

```
      A←1 2 3 4 5
      (+/1),(+/1 2),(+/1 2 3),(+/1 2 3 4),(+/1 2 3 4 5)
1 3 6 10 15
      +\A
1 3 6 10 15
      ×\A
1 2 6 24 120
```

Funktioargumenttinen kertymä on myös **operaattori**.

nousuindeksi ⍋

Monadisen nousuindeksifunktion ⍋*X* tulosvektorissa ovat numeerisen vektorin *X* **indeksit** alkioiden mukaisessa **nousevassa** suuruusjärjestyksessä.

```
        N←5 7 4 9 0
        ⍋N
5 3 1 2 4
```

Vektori lajitellaan nousevaan suuruusjärjestykseen nousuindeksifunktion tuloksen (indeksivektorin) ja hakasuljeindeksoinnin avulla.

```
        N[⍋N]
0 4 5 7 9
```

Nousuindeksiä voidaan käyttää myös **muiden** vektoreiden järjestämiseen halutun avainvektorin alkioiden mukaiseen suuruusjärjestykseen.

```
        'MARSU'[⍋N]
URMAS
```

laskuindeksi ⍒

Monadinen laskuindeksifunktio ⍒*X* toimii nousuindeksin tavoin, mutta sen tuloksena on nousevan sijasta **laskeva** suuruusjärjestys.

```
        ⍒N
4 2 1 3 5
        N[⍒N]
9 7 5 4 0
        'KAIVO'[⍒N]
VAKIO
```

heijastus (peilaus), kierto ⌽

Monadinen heijastusfunktio ⌽*X* vaihtaa vektorin *X* alkioiden järjestyksen päinvastaiseksi.

```
        ⌽'TALOUS'
SUOLAT
        ⌽N
0 9 4 7 5
```

Dyadisella kiertofunktiolla *N*⌽*V* kierretään vektorin *V* alkioita kokonaisluvun *N* itseisarvon verran joko vasempaan (alkupäätä eli origoa päin, *N*>0) tai oikealle (*N*<0).

```
        3⌽N
9 0 5 7 4
        ¯3⌽N
4 9 0 5 7
        2⌽'RIESKA'
ESKARI
        ¯4⌽'KURAUS'
RAUSKU
```

APL-ohjelmat

otsikkorivi

Muoto (hakasulkeissa olevat osat voivat puuttua, aaltosulkeissa olevat voivat esiintyä `0,...,N` kertaa):

`[tulosmuuttuja←] [v_arg] nimi [o_arg] {;lokaalimuuttuja}`

Jos funktiolla on vain yksi argumentti, se on oikeanpuoleinen argumentti (`o_arg`). Paikalliset muuttujat (**lokaali**muuttujat) ovat käytössä vain ohjelman sisällä. Jos lokaalimuuttujan arvoa muutetaan, ei minkään ohjelman ulkopuolisen samannimisen muuttujan arvo muutu. Muuttujat, jotka eivät ole paikallisia, ovat **globaaleja**. Globaalimuuttujat ovat käytettävissä (näkyvissä) sekä funktion sisä- että ulkopuolella, paitsi jos jokin lokaalimuuttuja on samanniminen. Tällöin lokaalimuuttujan nimi peittää *(shadows)* vastaavan globaalin nimen funktion suorituksen ajaksi. Globaalimuuttujan arvo **ei** riipu samannimisen lokaalimuuttujan arvosta.

Argumentit ovat paikallisia muuttujia, jotka saavat alkuarvoikseen ohjelman kutsussa esiintyvät arvot. Ohjelmia voidaan käyttää **rekursiivisesti** eli ne voivat kutsua itseään.

rivien numerointi

Otsikkorivin numero on `0` (nolla). Muut rivit ajatellaan numeroiduiksi juoksevasti `1,2,3,...` Rivin alussa oleva kaksoispisteeseen päättyvä nimi on paikallinen muuttuja (riviotsikko, nimiö) *(label)*, jonka arvo on sama kuin kyseisen ohjelmarivin numero. Nimiön arvoa **ei** voi sijoitusfunktiolla muuttaa.

hyppykäsky →

Hyppykäskyn `→L` *(branch, goto, jump)* vaikutuksesta (skalaaria käsitellään yksialkioisena vektorina):

- siirrytään seuraavalle riville, jos `L` on tyhjä vektori
- poistutaan ohjelmasta, jos `L[1]=0` (tai muu kokonaisluku joka **ei** ole funktion rivinumero)
- muutoin suoritus jatkuu riviltä, jonka rivinumero on `L[1]`
- niladinen hyppy *(escape, abort)* → keskeyttää **kaikkien** ohjelmien suorituksen.

kommentti (selite, huomautus) ⍝

Ohjelmarivillä symboli ⍝ (**ei** tekstivektorin osana) tulkitaan kommentiksi *(comment)* merkistä rivin loppuun.

HUOM: Jotkut APL-tulkit vaativat kommentit omiksi riveikseen.

esimerkkejä

```
[0]    Z←KERTOMA N
[1]     Z←1                        ⍝ yhden/nollan kertoma
[2]     →(N≤1)/0                   ⍝ joko kaikki?
[3]     Z←N×KERTOMA N-1            ⍝ rekursiivinen kutsu

[0]    Z←M SYT N
[1]    ⍝ Suurin yhteinen tekijä Euklideen menetelmällä
[2]    ⍙:Z←M                       ⍝ paluuarvo
[3]     M←M|N                      ⍝ laske jakojäännös
[4]     N←Z                        ⍝ vaihda päittäin
[5]     →(M≠0)⍴⍙                   ⍝ ellei tasan, uudestaan

       40 SYT KERTOMA 4
8
```

Hieman virheilmoituksista

Jos APL-tulkki kohtaa lausekkeen, jota se ei voi suorittaa, keskeytetään ohjelman suoritus. Ruudulle tulostuvassa kolmirivisessä virheviestissä on virhetyypistä riippuva lakoninen virheteksti, virhelauseke (ohjelmarivi) ja `^`-merkki, jolla osoitetaan se kohta johon suoritus pysähtyi.

Esimerkiksi:

```
         5÷0
DOMAIN ERROR                                  ⍝ määrittelyaluevirhe
         5÷0
         ^
```

Tavallisimpia virheilmoituksia ovat:

DOMAIN ERROR	- argumentti ei kuulu määrittelyalueeseen (esimerkiksi nollalla jako)
INDEX ERROR	- indeksiviittaus sääntiön ulkopuolelle
LENGTH ERROR	- argumenttien pituudet eivät täsmää
RANK ERROR	- argumenttien ulottuvuudet eivät täsmää
SYNTAX ERROR	- väärämuotoinen APL-lauseke
VALUE ERROR	- viittaus muuttujaan tai funktioon, jota ei ole olemassa
WS FULL	- työtilan muisti on täyttynyt.

Virhetilanne (joissakin tulkeissa myös istunnossa tehdyt virheet) rekisteröidään APL:n **tilanilmaisimeen** (virhetilannepino, tilaindikaattori) *(State Indicator).* Ohjelman suorituksen keskeydyttyä voi virhetilanteen poistaa (korjata muuttujat, näppäilyvirheet tms.) ja **jatkaa suoritusta** siitä, mihin se keskeytyi.

Keskeytyneen ohjelman saa jatkumaan korjauksen jälkeen hyppäämällä suoraan virheriville, mihin voi käyttää lauseketta `→⎕LC`. Tuo `⎕LC` *(Line Counter)* viittaa järjestelmämuuttujavektoriin, jonka ensimmäisenä alkiona on kulloisenkin suoritettavan ohjelmarivin numero. Järjestelmämuuttujat ja -funktiot alkavat **aina** ⎕-merkillä.

Työtilan tilanilmaisimen saa esille järjestelmäkomennolla `)SI`. Tilapinon saa tyhjennetyksi (tilaa viemästä) järjestelmäkomennolla `)RESET`. Järjestelmäkomennot alkavat aina kaarisulkeella.

Järjestelmämuuttujiin, -funktioihin ja -komentoihin palataan tarkemmin tuonnempana.

Kontrollirakenteista

APL:n kontrollirakenteet (ehto- ja toistorakenteet) on nopeasti kerrottu: hyppynuolen lisäksi niitä **ei** ole. Hyvässä APL-koodissa ei juuri kontrollirakenteita tarvita, koska käsiteltävän tiedon järjestäminen rakenteeltaan ongelmanratkaisuun sopivaksi vähentää skalaarikielille tyypillisten silmukoiden tarvetta.

Ehto- ja toistorakenteita on tarvittaessa helppo muodostaa. Esimerkiksi *N* kertaa suoritettava silmukka voidaan tehdä vaikkapa näin:

```
 I←0                          ⍝ kierroslaskuri
⍙0:I←I+1                      ⍝ silmukan alkuosoite
 →(N<I)/⍙1                    ⍝ joko kaikki?

  ...                         ⍝ silmukassa suoritettava APL-koodi

 →⍙0                          ⍝ takaisin alkuun
⍙1:                           ⍝ jatko-osoite
```

Yksinkertainen **jos-niin** -ehtorakenne *(if-then)* esitellään myöhemmin suoritusfunktion ⍎ yhteydessä.

<u>Työtila ja kirjastot, tiedostoista</u>

APL:n käyttämät objektit talletetaan perinteisesti **työtiloihin** *(workspace)*. Työtilaa käsitellään APL-istunnossa *(session)*, jolloin käytettävissä ovat kaikki APL:n perusfunktiot sekä ne muuttujat ja funktiot, jotka käsiteltävään työtilaan on muokattu (editoitu) taikka kopioitu. Työtila sisältää siten **koko** työskentely-ympäristön, eli virhetilanteeseen keskeytyneen funktion ja sen mahdollisten kutsufunktioiden keskeytyshetken lokaalimuuttujat ovat myös läsnä. Tämä toisaalta mahdollistaa virheellisen työtilan talletuksen myöhempää selvittelyä varten, toisaalta kuluttaa työtilan käyttömuistia.

Työtilojen hallintaan on kourallinen järjestelmäkomentoja *(system command)*, jotka alkavat aina **oikealla** sulkeella.

Talletettu työtila **ladataan** käytettäväksi komennolla `)LOAD ttnimi`, missä `ttnimi` tarkoittaa työtilan (taltio)nimeä. Eri ympäristöissä työtilojen nimeämissäännöt vaihtelevat. Yleensä kahdeksanmerkkinen APL- ja skandimerkkejä sisältämätön nimi kelpaa.

Työtilan **nimen** saa esiin argumentittomalla komennolla `)WSID` *(**W**ork**s**pace **Id**entifier)*. Ennen talletusta on työtila nimettävä joko tallennuskomennossa tahi sitten komennolla `)WSID ttnimi`.

Työtila **talletetaan** joko argumentittomalla komennolla `)SAVE` tai sitten käskyllä `)SAVE ttnimi`. Tehdyn työn talletus on <u>täysin</u> käyttäjän kontolla. Kokenutkin käyttäjä tulee silloin tällöin ladanneeksi uuden työtilan vaikka entinen oli tallettamatta.

HUOM: APL **ei varoita** tallettamattomuudesta, päällekopioinnista tms. – käyttäjähän tietää, mitä tekee!

Työtilaan **kopioidaan** muista APL-työtiloista objekteja komennolla `)COPY ttnimi nimilista`. Jos nimilistaa ei käytetä, kopioidaan koko työtila. Työtilassa mahdollisesti olevat samannimiset objektit korvautuvat kopiotyötilan objekteilla. Suojauskopiokomentoa `)PCOPY` *(**P**rotected **Copy**)* käytettäessä samannimisiä objekteja ei kopioida.

Työtilan funktiot listataan komennolla `)FNS` *(**F**u**n**ction**s**)* ja muuttujat komennolla `)VARS` *(**Var**iable**s**)*.

Työtilatiedoston voi poistaa **pysyvästi** komennolla `)DROP ttnimi`.

Työtilan objekteja voi poistaa käskyllä `)ERASE nimilista`.

Tyhjä työtila alustetaan komennolla `)CLEAR`. Tämä poistaa **kaikki** objektit ja virhetilanteet muistista.

Työtiloja voidaan järjestelmästä riippuen tallettaa eri **kirjastoihin**, joihin yleensä viitataan kirjastonumerolla. Eri APL-tulkkien kirjastointitavat poikkeavat toisistaan merkittävästi.

IBM:n APL:issä on perinteisesti käytössä kolmen tyyppisiä kirjastoja: julkisia, yksityisiä ja työryhmäkirjastoja. Esimerkiksi kirjastossa `1` on yleensä yleistyökaluja sisältävä työtila `UTILITY`, joka ladataan käyttöön käskyllä `)LOAD 1 UTILITY`.

APL-tulkit kykenevät nykyään yleensä käyttämään isäntäkoneensa tiedostojärjestelmää suoraan, jolloin kirjastoviitteiden asemesta voi työtiloihin viitata suoraan hakemistorakenteen mukaisilla nimillä.

APL-istunto lopetetaan enemmittä kyselyittä käskyllä `)OFF`.

Tiedostojen suoraan käsittelyyn ei perinteisesti ole standardoituja APL-välineitä. IBM-APL:issä tiedostoja (kuten muitakin APL:n ulkopuolisia resursseja) käsitellään **yhteismuuttujilla** *(shared variables)*. Useissa muissa APL-tulkeissa (mm. APL*PLUS, Dyalog APL, APLX, Sharp APL) on tiedostojen käsittelyä helpottavia järjestelmäfunktioita *(system functions)*, joilla voidaan suoraan käyttää joko APL-komponenttitiedostoja (*component files*) tahi muita tiedostoja *(native files)*.

Sääntiöt

Sääntiö *(array)* on suorakulmaisesti järjestettyjen homogeeninen (samantyyppisten alkioiden) joukko. Sääntiön rakenteen määrittävät sen **ulotteisuus** (aste, kertaluku) ja **koko**.

Sääntiön ulotteisuus *(rank)* on sen ulottuvuuksien (suuntien, dimensioiden) lukumäärä (yleensä enintään 64):

- yksiulotteinen sääntiö = **vektori** (alkiojono; yksi suunta) *(vector)*
- kaksiulotteinen sääntiö = **matriisi** (taulukko; kaksi suuntaa) *(matrix)*
- kolmiulotteinen sääntiö = **kuutio** (kolme suuntaa) *(cube, three-dimensional array)*
- nollaulotteinen sääntiö = **skalaari** (**ei** suuntia) *(scalar)*.

Sääntiön koolla (koko- eli koordinaattivektorilla) *(shape)* tarkoitetaan sen alkioiden lukumäärää eri ulottuvuuksien (akseleiden) suunnissa. Jokainen sääntiö on suorakulmainen (ortogonaalinen) eli alkioiden lukumäärä yhdessä suunnassa on riippumaton alkioiden määristä muissa suunnissa.

Esimerkiksi matriisin jokainen rivi on aina yhtä pitkä eli jokaisella rivillä on sama määrä sarakkeita eikä niiden lukumäärä riipu rivien määrästä.

Sääntiön **kokovektori** ilmaisee sääntiön alkioiden lukumäärät suunnittain. Kokovektori saadaan monadisella kokofunktiolla (`⍴K`); sääntiön ulotteisuus (suuntien määrä) on siten = kokovektorin koko (`⍴⍴K`).

Esimerkiksi `2 × 3 × 4` -matriisin kokovektori on `2 3 4` ja sen ulotteisuus on = `3`.

Jos jokin kokovektorin alkioista on nolla, on kyseessä **tyhjä** sääntiö *(empty array)*.
Skalaarin koko on tyhjä vektori ja sen ulotteisuus on = `0`.

Sääntiön alkioiden indeksointi eri suunnissa alkaa yleensä ykkösestä, mutta se voidaan myös muuttaa alkamaan nollasta asettamalla indeksialku-järjestelmämuuttujan `⎕IO` *(Index Origin)* arvo nollaksi. Tässä oppaassa indeksointi alkaa kuitenkin aina ykkösestä, ellei erikseen toisin mainita.

Yli kaksiulotteiset sääntiöt tulostetaan tasoittain kaksiulotteisina osamatriiseina. Eri ulottuvuustasojen rajalle tulostetaan aina yksi ylimääräinen tyhjä rivi.

Joidenkin funktioiden toimintaan (arvoalueeseen tai tulokseen) vaikuttavat järjestelmämuuttujat ilmaistaan funktioesittelyn otsikossa merkinnällä, jossa kyseiset järjestelmämuuttujat luetellaan aaltosulkeiden välissä, esimerkiksi `{⎕IO}`. Lisätietoa järjestelmämuuttujista ja -funktioista esitetään tuonnempana.

<u>*suuntamerkintä (suunnassa)*</u>   `f[S]X`   `{⎕IO}`

Useille funktioille (myös operaattoreilla johdetuille) voidaan osoittaa se suunta *(axis)*, jonka mukaan funktio sääntiössä suoritetaan. Suuntamerkintänä käytetään funktiomerkinnän **oikealle** puolelle sijoitettavia hakasulkeita, joiden väliin haluttu suunta sijoitetaan. Suunnan lukuarvo riippuu indeksialun arvosta.

Esimerkiksi matriisin `M` sarakesummat saadaan lausekkeella `+/[1]M` (tai `+/[0]M` jos `⎕IO=0`). Yleensä funktioiden suorituksen oletussuuntana on sääntiön **viimeinen** suunta, joten matriisin rivisummat saa joko lausekkeella `+/[2]M` tahi suoraan `+/M`. Joistakin funktiosymboleista on vielä vaakaviivallinen muunnos, jolla kyseinen funktio suoritetaan oletusarvoisesti **ensimmäisessä** suunnassa. Esimerkiksi matriisin sarakesummat saadaan suoraan lausekkeella `+⌿M`. Suuntamerkintää käytettäessä molemmat merkinnät ovat keskenään samanarvoiset (`+/[2]M ←→ +⌿[2]M`).

Sekafunktioiden yleiset muodot

Käytetään jatkossa seuraavia merkintöjä ja esimerkkimuuttujia:

S skalaari
V vektori
M matriisi
K kaikki sääntiöt
⍺ , ⍵ skalaarifunktioita
⋄ rivinerotin (timantti) lausekkeiden yhdistelyyn (**ei** mukana kaikissa APL-tulkeissa)
←→ lausekkeiden vastaavuus (**ei** oikea APL-ilmaisu)
`VEK ←→ 2 13 7 5 1` (vektori)
`MAT ←→ 3 4⍴1 2 3 4,5 6 7 8,9 10 11 12` (lukumatriisi)
`TXT←4 4⍴'OSATTULOOLATTASO'` (tekstimatriisi).

indeksointi (hakasuljeindeksointi) `K0[K1;..;Kn]` {⎕IO}

Indeksoinnin *(bracket indexing)* tuloksena on sääntiö, joka on koottu `K0`:sta indeksisääntiöillä `K1..Kn`. Indeksilausekkeessa tulee hakasulkeiden välissä olla `¯1+⍴⍴K` puolipistettä eli **jokaisen** suunnan indeksointi tulee antaa erikseen. Indeksien sallitut arvot ovat yhdestä `K0`:n kokovektorin vastaavan alkion arvoon (kun `⎕IO=1`). Esimerkiksi `2 × 3`-matriisilla toisen indeksin alkioiden tulee olla arvoltaan `1` , `2` tai `3` (jos `⎕IO=0`, sallitut arvot ovat `0` , `1` ja `2`).

Indeksisääntiö `Ki` edustaa `K0`:n `i`:nnettä suuntaa; sen alkiot ilmaisevat, missä järjestyksessä vastaavan suunnan alkioita poimitaan tulossääntiötä luotaessa. `Ki` voidaan aina muodostaa sopivalla APL-lausekkeella. Jos `Ki` on skalaari, puuttuu tulossääntiöstä vastaava `K0`:n suunta, muulloin tulossääntiössä `K0`:n `i`:nnettä ulottuvuutta vastaa yhtä monta ulottuvuutta kuin niitä `Ki`:ssä on. Jos `Ki` puuttuu, tarkoitetaan **kaikkien** vastaavan indeksin arvojen muodostamaa vektoria (`⍳(⍴K0)[i]`).

Tulokselle pätee: `⍴K0[K1;..;Kn] ←→ (⍴K1),..,⍴Kn` (puuttuvalle `Ki`:lle: `⍴Ki ←→ (⍴K0)[i]`).

```
          VEK[2]
13                          ⍝ skalaari, ⍴VEK[2] ←→ ⍴2
          VEK[4 3 2 1 2]
5 7 13 2 13
          'POREILU'[1+⌽⍳5]
LIERO
          'ABCDEFGHIJKLM'[MAT]
ABCD
EFGH
IJKL
          MAT[1 3;3 2 1]    ⍝ ⍴MAT[1 3;3 2 1] ←→ (⍴1 3),(⍴3 2 1) ←→ 2 3
3  2  1
11 10 9
          MAT[2;]           ⍝ ←→ MAT[2;1 2 3 4]
5 6 7 8
          MAT[;1]           ⍝ ⍴MAT[;1] ←→ (⍴1 2 3),(⍴1) ←→ ,3
1 5 9
          MAT[;⍳1]          ⍝ ⍴MAT[;⍳1] ←→ (⍴1 2 3),(⍴,1) ←→ 3 1
1
5
9
          ⍴MAT[2 3 4⍴⍳3;5 6 7⍴⍳4]
2 3 4 5 6 7
```

koko (dimensio, muoto) ρ*K*

Kokofunktion *(shape)* tuloksena on argumentin *K* koordinaatti**vektori** (kokovektori).

```
          ρVEK
5                             ⍝ ↔ ,5
          ρρVEK
1
          ρMAT
3 4
          0 0ρMAT             ⍝ ei tulostu edes yhtä tyhjää riviä
```

Skalaariargumentin koko on **tyhjä vektori**.

```
          ρ'A'                ⍝ skalaari!
                              ⍝ ↔ 0ρ0
          ρ''                 ⍝ tyhjä vektori
0                             ⍝ ↔ ,0
```

HUOM: Yleinen APL-käyttäjän virhe on sekoittaa yksialkioinen vektori ja skalaari keskenään; kummallakin voi olla sama sisältö, mutta eri **rakenne** (ts. vektori **koostuu** skalaareista).

koonti (kokoa, muotoile, strukturoi) *V*ρ*K*

Koonnin *(reshape)* tuloksena on sääntiö, jonka **kokovektoriksi** tulee vasen argumentti *V*. Tulossääntiön alkiot saadaan toistamalla *K*:n alkioita riveittäin (viimeinen indeksi vaihtuu nopeimmin) kunnes jokainen *V*:n määrittämistä paikoista on täytetty.

```
          3 4ρι12             ⍝ ↔ MAT
1  2  3  4
5  6  7  8
9 10 11 12
```

Jos vasempana argumenttina on skalaari, se vektoroidaan ennen koontia.

```
          A←7ρMAT ⋄ A         ⍝ A ↔ (,7)ρMAT
1 2 3 4 5 6 7
```

Tyhjällä vektorilla kokoaminen tuottaa **skalaarin**.

```
          1ρVEK               ⍝ ↔ (,1)ρVEK
2                             ⍝ ↔ ,2
          ''ρVEK              ⍝ ↔ (ι0)ρVEK
2                             ⍝ skalaari
```

Miksi skalaarin koko sitten on **tyhjä** vektori? Tarkastellaan lähemmin sääntiön kokoa ja ulotteisuutta:

```
          A←3 2 1ρ'W'         ⍝ ρA ↔ 0↓3 2 1 ↔ 3 2 1   ⍝ ρρA ↔ ,3
          A←  2 1ρ'W'         ⍝ ρA ↔ 1↓3 2 1 ↔   2 1   ⍝ ρρA ↔ ,2
          A←    1ρ'W'         ⍝ ρA ↔ 2↓3 2 1 ↔    ,1   ⍝ ρρA ↔ ,1
          A←     'W'          ⍝ ρA ↔ 3↓3 2 1 ↔    ι0   ⍝ ρρA ↔ ,0
```

Skalaarin ulotteisuus (aste) on siis = 0, jolloin vastaavan kokovektorin on oltava tyhjä.
Sääntiön ulotteisuuden voi yksinkertaisissa tapauksissa mieltää geometrisesti: siinä missä kolmi-, kaksi- ja yksiulotteiset sääntiöt vastaavat kuutiota, tasoa ja viivaa, vastaa skalaari yhtä **pistettä** jolla ei ulottuvuuksia ole.

otto $V{\uparrow}K$

Ottofunktion *(take)* tulossääntiö saadaan ottamalla K:n alkioista ohjausvektorin V osoittama viipale (osasääntiö) **mukaan**. Kukin V:n alkio ilmaisee, kuinka monta sen suuntaista tasoa K:sta valitaan.

Negatiiviselle V:n alkiolle otetaan tasot suunnan loppupäästä. Jos tasoja valitaan enemmän kuin niitä K:ssa on, ajatellaan K:n sisältävän ylimääräisiä tasoja, jotka kaikki ovat K:n tyypin mukaan joko nollia tai välilyöntejä. V:ssä on oltava yhtä monta alkiota kuin K:n kokovektorissa (⍴V ←→ ⍴⍴K).

```
        2 3↑MAT
1 2 3
5 6 7
        2 ¯6↑MAT             ⍝ ⍴V↑K ←→ |V
0 0 1 2 3 4
0 0 5 6 7 8
        ¯7↑'ABCD'
   ABCD                      ⍝ alkuun kolme välilyöntiä
```

pudotus (poisto, jättö) $V{\downarrow}K$

Pudotusfunktio *(drop)* toimii kuten ottofunktio, mutta vasen argumentti V määrittää nyt **pois** jätettävät tasot.

```
        ¯1 2↓MAT             ⍝ ⍴V↓K ←→ 0⌈(⍴K)-|V
3 4
7 8
        1 1↓¯1 2↓MAT
8                            ⍝ 1 1 -kokoinen matriisi!
        A←2 5 8 12 19 31
        (1↓A)-¯1↓A           ⍝ viereisten alkioiden erotukset
3 3 4 7 12
```

jäsenyys (joukkoon kuuluminen) $K1{\in}K2$ {⎕CT}

Jäsenyysfunktion *(member of)* tuloksena on `K1`:n muotoinen looginen sääntiö, jossa `K2`:een sisältyvien alkioiden arvo on `1`, muiden `0`.

```
        VEK∊MAT
1 0 1 1 1
        MAT∊VEK
1 1 0 0
1 0 1 0
0 0 0 0
```

jonoutus (vektorointi) $,K$

Jonoutusfunktion *(ravel)* tuloksena on vektori, jonka alkioina ovat K:n alkiot **riveittäin** (viimeinen indeksi vaihtuu nopeimmin). Jonoutuksella voi skalaarista tehdä yksialkioisen vektorin.

```
        ,MAT
1 2 3 4 5 6 7 8 9 10 11 12
        ⍴,'A'
1
```

liitos (kerrostus) K1,[S]K2   K1,K2 {⎕IO}

Liitoksen *(catenate)* tulossääntiössä on `K1`:n ja `K2`:n alkiot liitetty toisiinsa suunnassa `S`. Suuntamerkinnän puuttuessa on oletuksena **viimeinen** suunta. Argumenttien on oltava samaa tyyppiä (lukuja tai merkkejä). Jos `K1` ja `K2` ovat skalaareita tai vektoreita, on tuloksena **vektori**, jossa ovat argumenttien alkiot peräkkäin.

```
         VEK,100 200
2 13 7 5 1 100 200
```

Jos `S` on kokonaisluku ja argumenteilla on yhtä monta suuntaa, liitos tehdään suunnassa `S`. Tuloksen kokovektorin `S`:s alkio on `K1`:n ja `K2`:n vastaavien alkioiden summa, ja kokovektorin muiden alkioiden on oltava molemmilla argumenteilla **yhtä suuret**.
Esimerkiksi jos `ρK1 ←→ 2 3 5` ja `ρK2 ←→ 2 6 5`, on `ρK1,[2]K2 ←→ 2 9 5`.

```
         MAT,MAT              ⍝ ρMAT,MAT ←→ ρMAT,[2]MAT ←→ 3,4+4
1  2  3  4 1  2  3  4
5  6  7  8 5  6  7  8
9 10 11 12 9 10 11 12
```

Jos `S` on kokonaisluku ja toisella argumentilla on **yksi** suunta vähemmän kuin toisella, ajatellaan että siihen on lisätty uusi suunta `S` (kokovektoriin tulee uudeksi `S`:nneksi komponentiksi luku `1`), ja toimitaan kuten edellä.

```
         MAT,10 20 30         ⍝ ←→ MAT,[2]3 1ρ10 20 30
1  2  3  4 10
5  6  7  8 20
9 10 11 12 30
         ρ(2 3 4ρ1),[2]2 4ρ2           ⍝ ←→ ρ(2 3 4ρ1),[2]2 1 4ρ2
2 4 4
```

Jos `S` ei ole kokonaisluku, argumenttien kokovektorien on oltava identtiset. Kumpaankin argumenttiin ajatellaan lisätyksi uusi suunta `⌈S`, jota vastaavaksi kokovektorin komponentiksi tulee `1`, ja **kerrostus** *(laminate)* tehdään tässä uudessa suunnassa.

```
         (10×⍳5),[0.1]VEK   ⍝ ←→ (1 5ρ10×⍳5),[1]1 5ρVEK
10 20 30 40 50
 2 13  7  5  1
         (⍳3),[1.5]10 11 12
1 10
2 11
3 12
         (3 1ρ'OSO'),[2.9]3 1ρ'NEK'
ON

SE

OK
```

Jos toinen argumentti on skalaari, voi toinen olla mikä sääntiö tahansa. Tällöin skalaari ajatellaan korvatuksi sääntiöllä, jossa on yhtä monta ulottuvuutta kuin toisella argumentilla, kokovektorin liitossuunta-alkio `=1`, muut alkiot ovat samat kuin toisella argumentilla ja kaikkien alkioiden arvo = kyseisen skalaarin arvo.

```
         0,[1]MAT,100         ⍝ ←→ (1 5ρ0),[1]MAT,3 1ρ100
0  0  0  0   0
1  2  3  4 100
5  6  7  8 100
9 10 11 12 100
```

lukusarja ⍳*S* {⎕IO}

Monadinen lukusarjafunktio *(interval)* tuottaa positiivisesta kokonaisluvusta *S* vektorin ⎕IO,...,*S*-~⎕IO. Nolla-argumentilla saa tulokseksi usein tarpeellisen tyhjän vektorin (⍳0).

```
          ⍳4
1 2 3 4
          ⍴⍳0
0
          5-⍳4                  ⍝ ←→ ⌽⍳4
4 3 2 1
          ⍳1                    ⍝ ←→ ⎕IO vektorina
1
```

sijainti *V*⍳*K* {⎕IO ⎕CT}

Dyadisen sijaintifunktion *(index of)* tuloksena on *K*:n muotoinen sääntiö, jonka kukin alkio ilmaisee *K*:n vastaavan alkion sijainnin (indeksin) *V*:ssä. Alkion arvona on (⍴*V*)+⎕IO, jollei kyseistä *K*:n alkiota *V*:ssä ole.

```
          'TASKURAPU'⍳'SAKSET'
3 2 4 3 10 1
          VEK⍳MAT
5 1 6 6
4 6 3 6
6 6 6 6
```

satunnaisluku (otanta takaisinpanolla) ?*S* {⎕IO ⎕RL}

Monadinen satunnaislukufunktio *(roll)* määrittää positiiviselle kokonaisluvulle *S* (<2*31) satunnaisluvun joukosta ⍳*S*.

Yhden kortin vetoa jokerittomasta korttipakasta voi simuloida vaikka seuraavasti:

```
          ('♡◇♤+'[?4]),'1234567890JQK'[?13]
♤Q                              ⍝ ..patarouva tällä kertaa
```

satunnaisotos (otanta ilman takaisinpanoa) *S*1?*S*2 {⎕IO ⎕RL}

Dyadinen satunnaisotosfunktio *(deal)* tuottaa vektorin, jossa on *S*1 kappaletta satunnaisia **eri suuruisia** kokonaislukuja joukosta ⍳*S*2.

*S*1 ja *S*2 ovat positiivisia kokonaislukuja ja *S*1≤*S*2<2*31.

```
          7?39                  ⍝ lottonumeroarvonta
1 2 3 4 5 6 7                   ⍝ ..todennäköisyys aika pieni
          ?1                    ⍝ ←→ ⎕IO skalaarina
1
```

HUOM: Jokainen viittaus ?-satunnaislukufunktioihin muuttaa sivutuotteena myös satunnaislukusiemenjärjestelmämuuttujan ⎕RL *(Random Link)* arvoa.

supistus (typistys, tiivistys) *V*/[*S*]*K* *V*/*K* *V*⌿*K*

Supistusfunktion *(compress)* tulossääntiössä ovat mukana ne oikean argumentin *K* suunnassa *S* olevat tasot, joita vastaa loogisen vektorin *V* **ykkösalkio**. Jos *V*:n kaikki alkiot ovat nollia, on tuloksena tyhjä sääntiö.

Joko ⍴*V* ←→ (⍴*K*)[*S*] tai sitten *V* voi olla skalaari, joka ajatellaan toistetuksi riittävän monta kertaa. Suuntamerkinnän puuttuessa on oletuksena viimeinen suunta (*V*⌿*K*:lle ensimmäinen).

```
          1 0 0 1/MAT          ⍝ ←→ 1 0 0 1/[2]MAT ←→ 1 0 0 1⌿[2]MAT
1  4
5  8
9 12
          0 1 0⌿MAT            ⍝ ←→ 0 1 0/[1]MAT
5 6 7 8
          A←'RUISLINNUN LAULU KORVISSANI, TÄHKÄPÄIDEN PÄÄLLÄ TÄYSI-KUU'
          (A∊' ,-')/⍳⍴A        ⍝ kiinnostavien merkkien indeksit
11 17 28 29 41 48 54
          N←1 4 0 2 5 9 0 4 0 6 5 9
          (2|N)/N              ⍝ parillisten lukujen poisto vektorista
1 5 9 5 9
```

lavennus (harvennus) *V*\[*S*]*K* *V**K* *V*⍀*K*

Lavennuksen *(expand)* tulossääntiössä on *K*:hon lisätty sen *S*:nnessä suunnassa ylimääräisiä *K*:n tyypin mukaisia täytealkioita (nollia tai välilyöntejä) sisältävä taso jokaista loogisen vektorin *V* **nolla-alkiota** kohden.

Edellytetään että +/*V* ←→ (⍴*K*)[*S*]. Lavennuksessa **ei** käytetä skalaarilaajennusta. Suuntamerkinnän puuttuessa on oletuksena viimeinen suunta (*V*⍀*K*:lle ensimmäinen).

```
          (17⍴1 0)\'HARVENNUS'
H A R V E N N U S
          1 0 1 1⍀MAT          ⍝ ←→ 1 0 1 1\[1]MAT ←→ 1 0 1 1⍀[1]MAT
1  2  3  4
0  0  0  0
5  6  7  8
9 10 11 12
```

Lavennusfunktiota voi käyttää tietotyypin tutkintaan:

```
          0\⍳0                 ⍝ argumentti saa olla tyhjä vektori:
0                              ⍝  numeerinen -> 0
          0\''
                               ⍝  merkkitieto -> ' '
          0=0\0⍴'TEKSTIÄ'      ⍝  => teksti- tai numerotiedon
0
          0=0\0⍴MAT            ⍝     tyyppitarkastus!
1
```

HUOM: Tyyppitarkastuksen voi yhtä hyvin tehdä myös ottofunktiolla:

```
          0=1↑0⍴'TEKSTIÄ'
0
          0=1↑0⍴MAT
1
```

nousuindeksi ⍋*V*

{⎕IO}

Monadisen nousuindeksin *(grade up)* tuloksena on vektori, jossa numeerisen argumenttivektorin *V* indeksit ovat sen alkioiden mukaisessa **nousevassa** suuruusjärjestyksessä.

```
         ⍋VEK
5 1 4 3 2
         VEK[⍋VEK]
1 2 5 7 13
         A←27 4 ¯1 2 15 2 ◇ ⍋⍋A          ⍝ nousevat sijaluvut
6 4 1 2 5 3
```

laskuindeksi ⍒*V*

{⎕IO}

Laskuindeksi *(grade down)* toimii kuten nousuindeksi, mutta suuruusjärjestys on nyt **laskeva**.

```
         ⍒VEK
2 3 4 1 5
         VEK[⍒VEK]
13 7 5 2 1
         ⍋⍒A                             ⍝ laskevat sijaluvut
1 3 6 4 2 5
```

heijastus (peilaus) ⌽[*S*]*K* ⌽*K* ⊖*K*

Heijastusfunktion *(reverse)* tulossääntiössä ovat *K*:n tasot suunnassa *S* **päinvastaisessa** järjestyksessä kuin *K*:ssa. Suuntamerkinnän puuttuessa on oletuksena viimeinen suunta (⊖*K*:lle ensimmäinen).

```
         ⌽MAT                            ⍝ ←→ ⌽[2]MAT ←→ ⊖[2]MAT
 4  3  2 1
 8  7  6 5
12 11 10 9
         ⊖MAT                            ⍝ ←→ ⌽[1]MAT ←→ ⊖[1]MAT
9 10 11 12
5  6  7  8
1  2  3  4
         TXT
OSAT
TULO
OLAT
TASO
         ⌽TXT
TASO
OLUT
TALO
OSAT
         ⊖TXT
TASO
OLAT
TULO
OSAT
```

kierto K1 ⌽[S]K2   K1⌽K2   K1⊖K2

Dyadisen kiertofunktion *(rotate)* tuloksena on K2:n muotoinen sääntiö, jossa kutakin suunnan S mukaista tasoa on kierretty kokonaislukusääntiön K1 vastaavan alkion määräämien positioiden verran. Suuntamerkinnän puuttuessa on oletuksena viimeinen suunta (K1⊖K2:lle ensimmäinen).

Positiiviset K1:n alkiot kierrättävät vastaavaa tasoa origoon päin, negatiiviset origosta pois päin. Skalaarina K1 tulkitaan sopivan muotoiseksi sääntiöksi, jonka kaikkien alkioiden arvo = skalaarin arvo. Jos K2:n kokovektorista poistetaan S:s alkio, saatu vektori on **identtinen** K1:n kokovektorin kanssa. Esimerkiksi jos ⍴K2 ←→ 5 3 4 ja S←2, on oltava ⍴K1 ←→ 5 4.

```
         1⌽VEK                  ⍝ ←→ ¯4⌽VEK
13 7 5 1 2
         1⌽TXT                  ⍝ ←→ 1 1 1 1⌽A ←→ 1⌽[2]A
SATO
ULOT
LATO
ASOT
         1 3 2 4⊖TXT            ⍝ ←→ 1 3 2 4⌽[1]A
TAAT
OSSO
TUAT
OLLO
         ¯1 0 1 2⊖MAT           ⍝ ←→ ¯1 0 1 2⌽[1]MAT
9  2  7 12
1  6 11  4
5 10  3  8
         B←2 3 5⍴⍳30 ⋄ B
 1  2  3  4  5
 6  7  8  9 10
11 12 13 14 15

16 17 18 19 20
21 22 23 24 25
26 27 28 29 30
         1⌽[1]B                 ⍝ ←→ 1⊖B
16 17 18 19 20
21 22 23 24 25
26 27 28 29 30

 1  2  3  4  5
 6  7  8  9 10
11 12 13 14 15
         C←2 3⍴⍳6 ⋄ C
1 2 3
4 5 6
         C⌽[3]B                 ⍝ ←→ C⌽B ←→ C⊖[3]B
 2  3  4  5  1
 8  9 10  6  7
14 15 11 12 13

20 16 17 18 19
21 22 23 24 25
27 28 29 30 26
```

muotoilu (tulostusasu) ⍕*K*

{⎕PP}

Monadinen muotoilufunktio *(format, thorn)* muuttaa argumenttinsa samanlaiseksi tekstisääntiöksi kuin millaisena tämä tulostuisi ruudulle.

Tulossääntiölle on voimassa: `ρρ⍕K ←→ ,1⌈ρρK`.

```
        N←2 2 5ρ0.1×⍳20 ⋄ ⍕N
0.1 0.2 0.3 0.4 0.5
0.6 0.7 0.8 0.9 1

1.1 1.2 1.3 1.4 1.5
1.6 1.7 1.8 1.9 2
        ρ⍕N
2 2 19
```

Muotoilun avulla voi esittää sekä tekstiä että numeerista dataa yhdessä.

```
        'PII = ',⍕○1
PII = 3.141592654
```

esimerkkimuotoilu V⍕K

{⎕PP}

Dyadisen muotoilufunktion *(format by specification)* vasempana argumenttina on yksi tai useampi sääntiön sarakkeiden (viimeisen suunnan) muotoilua ohjaava **lukupari**. Lukuparit liittyvät oikean argumentin lukuihin siten, että parin edellinen jäsen määrittää kirjoitustilan leveyden ja jälkimmäinen desimaalien määrän.

Selkeyden vuoksi välilyönnit osoitetaan seuraavissa esimerkeissä alaviivalla (_).

```
        A←12.34 ¯23.456 1 ¯345.6789
        4 1 6 0 7 2 8 3⍕A           ⍝ ←→ (4 1,6 0,7 2,8 3)⍕A
12.3___¯23___1.00¯345.679
```

Jos lukuparin ensimmäinen jäsen on nolla, valitaan sellainen kirjoitustila, jolla luku **ei kosketa** naapuriin.

```
        6 1 0 0 0 2 0 3⍕A
__12.3_¯23_1.00_¯345.679
```

Jos vasempana argumenttina on vain yksi lukupari, se käsitetään kaikille **yhteiseksi**.

```
        8 1⍕A                       ⍝ ←→ 8 1 8 1 8 1 8 1⍕A
____12.3___¯23.5_____1.0__¯345.7
```

Yhteisen lukuparin ensimmäinen luku voidaan jättää kirjoittamatta, jolloin se tulkitaan nollaksi.

```
        1⍕A                         ⍝ ←→ 0 1 0 1 0 1 0 1⍕A
_12.3_¯23.5_1.0_¯345.7
```

Eksponenttimuotoinen (skaalattu) tulostus saadaan antamalla desimaalien lukumäärä negatiivisena. Skaalaimen itseisarvo osoittaa nyt **tulostustarkkuuden** (merkitsevien numeroiden lukumäärän).

```
        10 ¯4⍕A
_1.234E1__¯2.346E1___1.000E0__¯3.457E2__
```

suoritus ⍎*V*

Monadinen suoritusfunktio *(execute)* poistaa vektori- tai skalaariargumentiltaan tekstitieto-ominaisuuden.

```
         A←'5+3'
         ⍎A,'×',A                 ⍝ ↔ ⍎'5+3×5+3'
29
         M←2 4⍴'5  3','2 ¯1'
         ⍎,M                      ⍝ tekstimatriisista numerovektori
5 32 ¯1
         ⍎,M,' '                  ⍝ rivinvaihto!
5 3 2 ¯1
```

Tyhjän vektorin suoritus **ei** palauta tulosta, joten suoritusfunktiota voi käyttää supistusfunktion kanssa jos-niin -**ehtolausekkeen** *(if-then)* tavoin.

```
         ⍎(1<0)/'1+1'             ⍝ ehtolauseke, ei suoriteta
         ⍎(2|7)/'2+2'             ⍝ ehtolauseke, suoritetaan
4
```

transponointi (käännös) ⍉*K*

Transponoinnin *(transpose)* tulossääntiössä on *K*:n suunnat käännetty **päinvastaiseen** järjestykseen.

```
         ⍉MAT                     ⍝ ⍴⍉MAT ↔ ⌽⍴MAT ↔ 4 3
1 5  9
2 6 10
3 7 11
4 8 12
         ⍉TXT
OTOT
SULA
ALAS
TOTO
```

koordinaattien vaihto *V*⍉*K* {⎕IO}

Koordinaattien vaihdon *(dyadic transpose)* tulossääntiössä on *K*:n kokovektorin **järjestys** muutettu kokonaislukuvektorilla *V* ohjaten. *V* määrittää miksi tulossääntiön suunniksi *K*:n suunnat tulevat (⍴*V* ↔ ⍴⍴*K*). *V*:ssä voi olla **samoja** suuntia, jolloin tuloksena ovat näin määritellyt **diagonaalialkiot** (⍴⍴*V*⍉*K* ↔ ⌈/*V*).

V:n alkioiden tulee sisältää **kaikki** kokonaislukusarjan ⍳⌈/*V* luvut.
Transponoinnin ja koordinaattien vaihdon välillä on yhteys: ⍉*K* ↔ (⌽⍳⍴⍴*K*)⍉*K*.

Esimerkiksi kolmiulotteiselle sääntiölle *A*:

```
         Z←2 3 1⍉A => Z[I;J;K] ↔ A[J;K;I], ja
         Z←1 2 1⍉A => Z[I;J]   ↔ A[I;J;I].

         2 1⍉MAT                  ⍝ ↔ ⍉MAT
1 5  9
2 6 10
3 7 11
4 8 12
         1 1⍉MAT                  ⍝ ↔ MAT[1;1],MAT[2;2],MAT[3;3]
1 6 11
```

koodin avaus (tulkinta koodina) `K1⊥K2`

Koodin avauksen *(decode, base)* tuloksena on `K1`:n määrittämässä kannassa esitetyn `K2`:n arvo kymmenjärjestelmässä. Jokainen `K1`:n **viimeisen** suunnan mukainen vektori määrittää **kannan** eli ne luvut, joilla painotettuina kunkin `K2`:n **ensimmäisen** suunnan mukaisten vektorien alkiot lasketaan yhteen yksittäiseksi tulossääntiön alkion arvoksi.

Edellytetään, että `(¯1↑⍴K1) ←→ 1↑⍴K2`. Jos `K1`:n viimeinen (tai `K2`:n ensimmäinen) ulottuvuus on = `1`, se ajatellaan laajennetuksi vaadittavaan muotoon. Skalaariargumentti laajennetaan samalla tavoin.

Tulossääntiön kokovektori on muotoa `(¯1↓⍴K1),1↓⍴K2`.

```
          10⊥1 9 7 5          ⍝ ←→ 10 10 10 10⊥1 9 7 5
1975
          0 24 60⊥1 2 3       ⍝ yksi vrk, 2h ja 3min minuutteina
1563                          ⍝ (1×24×60)+(2×60)+3
```

HUOM: Nolla toimi tässä pituuden **täytealkiona**, koodin avauksessa sitä ei itse asiassa tarvittu!

```
          100⊥⌽19 10 1985     ⍝ päiväysvektorin koodaus yhdeksi luvuksi
19851019
          (2 3⍴⍳6)⊥3 5⍴100+⍳15
1035 1045 1055 1065 1075      ⍝ esim. 3888 ←→ (104×5×6)+(109×6)+114
3777 3814 3851 3888 3925

          A←3 2⍴'07','0A','FF'
          16⊥¯1+'0123456789ABCDEF'⍳⍉A
7 10 255                      ⍝ heksadesimaaliluvut desimaalisiksi
```

koodaus (esitys koodina) `K1⊤K2`

Koodauksen *(encode)* tuloksena on `K2`:n esitys `K1`:n määrittämässä kannassa. Kukin `K1`:n viimeisen suunnan mukainen vektori määrittää sen kannan, johon `K2`:n alkiot muunnetaan.

Tulossääntiön alkioiden arvot syntyvät lopusta alkuun **jakojäännöksinä**, ja ne ovat aina pienempiä kuin vastaavat koodivektorin alkiot. Koodatulla luvulla on yhtä monta alkiota kuin `K1`:n viimeisessä suunnassa, merkitsevämpiä alkioita **ei** oteta huomioon. Tulossääntiön kokovektori on muotoa `(⍴K1),⍴K2`.

```
          2 2 2 2 2 2⊤9
0 0 1 0 0 1
```

Nollalla koodaus antaa alkuperäisen luvun tai sen, mitä siitä on jäljellä.

```
          0⊤521
521
          0 24 60⊤1563
1 2 3
```

Huomaa, että seuraavissa esimerkeissä on havainnollisuuden takia tulosmatriisi käännetty.

```
          ⍉0 100 100⊤19961218 19950131 19590406
1996 12 18                    ⍝ VVVVKKPP-muotoisten päiväysten purku
1995  1 31
1959  4  6
          ⍉'0123456789ABCDEF'[1+16 16⊤7 10 255]
07                            ⍝ desimaaliluvut (<256) heksadesimaalisiksi
0A
FF
```

käänteismatriisi (domino) ⌹*M*

Monadisen dominofunktion *(matrix inverse)* tuloksena on kaksiulotteisen matriisin *M* (vasen) käänteismatriisi. Matriisin sarakkeiden on oltava toisistaan lineaarisesti riippumattomia.

Jos *M* on vektori, se tulkitaan yksisarakkeiseksi matriisiksi; skalaari tulkitaan `1 × 1` -matriisiksi.

```
        A←3 3⍴1 2 1,3 4 7,1 0 3 ⋄ A
1 2 1
3 4 7
1 0 3
        ⌹A
 3   ¯1.5  2.5
¯0.5  0.5 ¯1
¯1    0.5 ¯0.5
        ⌹5                 ⍝ ←→ ÷5
0.2
```

`(⌹M)+.×M` ←→ **yksikkömatriisi** mahdollisin pyöristysvirhein (`+.×` on **matriisitulo**).

```
        (⌹A)+.×A           ⍝ ks. sisätulo-operaattori
1 0 0
0 1 0
0 0 1
```

yhtälöryhmän ratkaisu (matriisijako) `M1⌹M2`

Dyadisen dominofunktion *(matrix divide)* tuloksena on yhtälöryhmän tai -ryhmien `M1 = M2 +.× X` ratkaisu **pienimmän neliösumman** menetelmällä. `M2`:n sarakkeiden on oltava lineaarisesti riippumattomia ja kummallakin argumentilla on oltava yhtä monta riviä, eli `(¯1↑⍴M1)` ←→ `¯1↑⍴M2`.

Esimerkiksi yhtälöryhmä $A +.× X_1, X_2, X_3 = 1\ 1\ 0$, eli

$$
\begin{aligned}
1×X_1 + 2×X_2 + 1×X_3 &= 1\\
3×X_1 + 4×X_2 + 7×X_3 &= 1\\
1×X_1 + 0×X_2 + 3×X_3 &= 0
\end{aligned}
$$

voidaan ratkaista lausekkeella (poistetaan pyöristysvirheet muotoilemalla tulos yhdellä desimaalilla):

```
        1⍕1 1 0⌹A
1.5 0.0 ¯0.5
```

Siispä X_1=`1.5` , X_2=`0` ja X_3=`¯0.5`.

Usean yhtälöryhmän samanaikainen ratkaisu sujuu myös:

```
        B←⍉4 3⍴1 1 0,0 5 5,3 3 ¯1,2 3 2 ⋄ B
1 0  3 2
1 5  3 3
0 5 ¯1 2
        1⍕B⌹A
 1.5  5.0  2.0  6.5
 0.0 ¯2.5  1.0 ¯1.5
¯0.5  0.0 ¯1.0 ¯1.5
```

Operaattorit

Operaattorit vaikuttavat operandifunktioidensa suoritustapaan siten, että syntyy uusia (johdannais)funktioita.

pelkiste (reduktio) `α/[S]K` `α/K` `α⌿K`

Pelkistysoperaation *(reduce, reduction)* tuloksena on sääntiö, jossa `K`:n `S`:nnen suunnan vektorit on korvattu skalaareilla. Skalaarien arvot saadaan laskutoimituksesta, joka syntyy merkitsemällä vastaavan vektorin alkioväleihin **skalaarifunktio**merkintä `α`. Johdetun funktion argumentti `K` on numeerinen paitsi funktioilla `=` ja `≠`.

Jos `(⍴K)[S] ←→ 1`, **ei** laskutoimitusta suoriteta. Jos suuntamerkintä puuttuu, on oletuksena viimeinen suunta (`α⌿K`:lle ensimmäinen).

```
        ×/VEK                   ⍝ ←→ 2×13×7×5×1
910
        ∨/0 1 1 0
1
        +/MAT                   ⍝ ←→ +/[2]MAT ←→ +⌿[2]MAT
10 26 42
        +⌿MAT                   ⍝ ←→ +/[1]MAT ←→ +⌿[1]MAT
15 18 21 24
```

Tyhjälle argumentille `α/K` palauttaa pelkistefunktion **identtisyysalkion** (*identity element*), joka funktion argumenttina ollessaan antaa tulokseksi toisen argumentin, tai tällaisen puuttuessa virheilmoituksen.

```
        ×/⍳0                    ⍝ X ←→ X×1
1
        -/⍳0                    ⍝ X ←→ X-0
0
        */⍳0                    ⍝ X ←→ X*1
1
        ⍟/12345
12345
        ⍟/⍳0
DOMAIN ERROR                    ⍝ määrittelyaluevirhe
```

selaus (kertymä, kumulointi) `α\[S]K` `α\K` `α⍀K`

Selausoperaation *(scan)* tuloksena on `K`:n muotoinen sääntiö, jonka kukin alkio edustaa vastaavaa `S`:nnen suunnan `α`-pelkistettä sille `K`:n osasääntiölle, jolla kyseisessä suunnassa on alkioita vain tähän alkioon asti. Jos suuntamerkintä puuttuu, on oletuksena viimeinen suunta (`α⍀K`:lle ensimmäinen).

```
        ×\VEK
2 26 182 910 910
        ∨\0 1 1 0 1 1           ⍝ ensimmäisen ykkösen jälkeiset ykkösiksi
0 1 1 1 1 1
        <\0 1 1 0 1 1           ⍝ vain ensimmäinen ykkönen
0 1 0 0 0 0
        ≠\0 1 1 0 1 1           ⍝ juokseva pariteetti ;)
0 1 0 0 1 0
        +⍀MAT                   ⍝ ←→ +\[1]MAT ←→ +⍀[1]MAT
 1  2  3  4
 6  8 10 12
15 18 21 24
```

ulkotulo (taulukko) K1 ∘.ω K2

Ulkotulon *(outer product)* tulossääntiössä on alkio kutakin K1:n ja K2:n alkioiden muodostamaa paria kohden. Tulosalkion arvo saadaan suorittamalla parin alkioiden välillä **skalaarifunktio** ω.
Tuloksen kokovektori on muotoa (⍴K1),⍴K2. Tulossääntiön alkioille on voimassa identiteetti:
(K1 ∘.ω K2)[G;..;H;I;..;J] ←→ K1[G;..;H] ω K2[I;..;J].

```
        (⍳5)∘.×⍳6           ⍝ kertotaulu
1  2  3  4  5  6
2  4  6  8 10 12
3  6  9 12 15 18
4  8 12 16 20 24
5 10 15 20 25 30
        (⍳4)∘.≥⍳4           ⍝ alakolmiomatriisi
1 0 0 0
1 1 0 0
1 1 1 0
1 1 1 1
        3⍕ ÷(⍳5)∘.+⍳5       ⍝ 5x5-Hilbert-matriisi
 0.500 0.333 0.250 0.200 0.167
 0.333 0.250 0.200 0.167 0.143
 0.250 0.200 0.167 0.143 0.125
 0.200 0.167 0.143 0.125 0.111
 0.167 0.143 0.125 0.111 0.100
```

sisätulo (yhdiste) K1 α.ω K2

Sisätulon *(inner product)* tulossääntiössä kutakin K1:n viimeisen ja K2:n ensimmäisen suunnan mukaista vektoriparia V1,V2 vastaa tulossääntiössä alkio, jonka arvo on α/ V1 ω V2.

Skalaarifunktion ω argumenteiksi tulevissa vektoreissa on oltava yhtä monta alkiota. Jos sisätulon toisena argumenttina on vain yksi alkio, sitä toistetaan tarvittaessa skalaarilaajennussäännön mukaan.
Tuloksen kokovektori on muotoa (¯1↓⍴K1),1↓⍴K2. Tulossääntiölle on voimassa identiteetti:
(K1 α.ω K2)[G;..;H;I;..;J] ←→ α/ K1[G;..;H;] ω K2[;I;..;J].

```
        A←3 3⍴'OLIISOILO' ⋄ A
OLI
ISO
ILO
        A∧.='ISO'
0 1 0
        B←⍉7 3⍴'UNIASUOLOELIISOOMAILO' ⋄ B
UAOEIOI
NSLLSML
IUOIOAO
        A∧.=B
0 0 0 0 0 0 0
0 0 0 0 1 0 0
0 0 0 0 0 0 1
```

Tavanomaista matriisikertolaskua vastaa sisätulo M1 +.× M2.

```
        E+.×2 3 5 7         ⍝ esim. 51 ←→ +/1 2 3 4 × 2 3 5 7
51 119 187
```

Tärkeimpiä järjestelmämuuttujia ja -funktioita

APL:n järjestelmämuuttujien ja -funktioiden nimet alkavat ⎕-merkillä.

järjestelmämuuttujia:

⎕AI *(Account Information)* Käyttömäärävektori: käyttäjätunnus, kertyneet CPU- ja yhteysajat millisekunteina.

⎕AV *(Atomic Vector)* Koodivektori: käytettävissä olevat merkit vektorina (⍴⎕AV ←→ 256).

⎕CT *(Comparison Tolerance)* Vertailutarkkuus: oletusarvo 10E¯13, sallitut arvot 0..1.
Vertailtavat ovat yhtä suuria, jos (⎕CT×⌈/|X,Y)≥|X-Y.
Vaikuttaa perusfunktioihin: ⌈ ⌊ < > = ≠ ≤ ≥ ⍳ ∊.

⎕IO *(Index Origin)* Indeksialku (indeksien alin arvo): oletusarvo 1, sallitut arvot 0 ja 1.
Vaikuttaa indeksointiin ja funktioiden suoritussuuntaan [ ] sekä perusfunktioihin: ⍳ ⍋ ⍒ ⍉ ?.

⎕LC *(Line Counter)* Suoritettavan ohjelman/ohjelmien rivinumero(t).

⎕LX *(Latent Expression)* Työtilan **latauksessa** suoritettava APL-lauseke tekstivektorina; oletusarvo ''.

⎕PP *(Printing Precision)* Tulostustarkkuus (tulostuksessa käytettävien desimaalien määrä): oletusarvo 10, sallitut arvot 1..16. Vaikuttaa kirjoituspyyntöön ⎕ ja muotoilufunktioon ⍕.

⎕PW *(Printing Width)* Tulostusalueen leveys: oletusarvo 80, sallitut arvot 20..255.
Vaikuttaa kirjoituspyyntöön ⎕.

⎕RL *(Random Link)* Satunnaislukusiemen ?-funktioille. Tyhjän työtilan alkuarvo on 16807 (7*5).
Jokainen viittaus satunnaislukufunktioihin muuttaa aina ⎕RL:n arvoa.

⎕TS *(Time Stamp)* Aikaleimavektori: vuosi, kuukausi, päivä, tunti, minuutti, sekunti, millisekunti.

⎕WA *(Workspace Available)* Vapaan työtilan koko tavuina.

järjestelmäfunktioita:

⎕CR *(Canonical Representation)* Merkkiesitys: tekstiargumentin funktionnimeä vastaava muotoiltu merkkimatriisi.

⎕DL *(Delay)* Viive: aiheuttaa skalaariargumenttinsa mittaisen sekuntiviipeen, palauttaa tarkan suoritusaika-arvon.

⎕EX *(Expunge)* Poisto: pyyhkii argumenttinsa (merkkimatriisi tai -vektori) ilmaisemat objektit pois työtilasta.

⎕FX *(Function Establishment)* Aktivointi: muuttaa argumenttitekstimatriisin funktioksi.

⎕NC *(Name Classification)* Nimiluokka: kertoo argumenttinsa (merkkimatriisi tai -vektori) objektiluokan (0/1/2/3 = ei käytössä/riviosoite/muuttuja/funktio).

⎕NL *(Name List)* Nimiluettelo: oikea argumenttiskalaari/vektori ilmaisee sen objektiluokan, josta etsitään (1= riviosoitteet, 2 = muuttujat ja 3 = funktiot); mahdollinen vasen argumentti ilmaisee millä kirjaimilla alkaviin nimiin rajoitutaan.

Tärkeimpiä järjestelmäkomentoja

APL:n järjestelmäkomentorivi alkaa aina **oikealla** sulkeella.
Komentorivillä **ei** saa olla mitään muuta kuin kyseinen komento parametreineen.

`)CLEAR`

Työtilan tyhjennys kaikista objekteista ja virhetilapinosta, nimeksi tulee `CLEAR WS` (vaihdettava ennen tallettamista).

`)COPY ttnimi {obj1 obj2 ..}`

Kopioi työtilaan työtilan `ttnimi` sisältö, tai jos objektilista `obj1,..` on annettu, kopioi vain nämä.

`)DROP ttnimi`

Poista työtilatiedosto `ttnimi`. Nimi on annettava täydellisenä kirjastoviitteineen.

`)ERASE obj1 {obj2 ..}`

Poista työtilasta objektit `obj1,...`

`)FNS {C}`

Listaa työtilan funktiot (jotka alkavat kirjaimilla `C..'Z'`) (`FunctioNS`).

`)LIB {ttlib}`

Listaa kirjaston `ttlib` työtilat (oletus = käynnistyskirjasto) (`LIBrary`).

`)LOAD ttnimi`

Työtilan `ttnimi` lataus muistiin (työtilassa olevat entiset tiedot menetetään).

`)OFF`

Istunnon päättäminen. **Ei** varmistuskyselyjä, muista tallettaa työsi!

`)PCOPY ttnimi {obj1 obj2 ..}`

Kuten `)COPY`, mutta kopioidaan vain ne objektit, joiden nimiä ei ole aktiivisessa työtilassa käytössä (`Protected COPY`).

`)RESET`

Siivoa virhetilapino.

`)SAVE {ttnimi}`

Työtilan talletus joko entisellä nimellä tai **uudelle** nimelle `ttnimi`.

`)SI`

Virhetilailmaisimen sisältö (`State Indicator`).

`)VARS {C}`

Listaa työtilan muuttujat (jotka alkavat kirjaimilla `C..'Z'`) (`VARiableS`).

`)WSID {ttnimi}`

Työtilan täysnimen tiedustelu tai asetus nimelle `ttnimi` (`WorkSpace IDentifier`).

HUOM: Järjestelmäkomentoja voi antaa tavallisesti vain istunnossa, ohjelmallisesti näiden käyttö ei kaikilla APL-tulkeilla ole mahdollista.

Idiomeja

APL-ohjelmoinnissa usein käytettäviä tiettyyn toimintoon sopivia funktioilmaisuja kutsutaan **idiomeiksi**. Seuraavassa on muutama yleisesti käytetty lyhyehkö idiomi (⎕IO=1):

```
0∊1↑0⍴V                                 ⍝ onko vektori numeerinen?
∧/(⍳⍴V)∊V                               ⍝ onko vektori permutaatiovektori?
∧/,K=1⍴K                                ⍝ ovatko sääntiön kaikki alkiot samoja?
⌊K+0.5                                  ⍝ pyöristys lähimpään kokonaislukuun (K≥0)
(K2≠0)×K1÷K2+K2=0                       ⍝ palauta nollalla jaettaessa nolla
0⊥V                                     ⍝ lukuvektorin viimeinen luku skalaarina
1⊥V                                     ⍝ lukuvektorin summa (+/V)
((1+⌊10⍟S)⍴10)⊤S                        ⍝ kokonaislukuskalaarin (>0) purku numeroiksi
0 1⊤K                                   ⍝ luvun (>0) kokonais- ja desimaaliosat
V∘.+,0                                  ⍝ numeerisesta vektorista pystymatriisi (((⍴V),1)⍴V)
(+/V)÷⍴V                                ⍝ aritmeettinen keskiarvo
(×/V)*÷⍴V                               ⍝ geometrinen keskiarvo
(⍴V)÷+/÷V                               ⍝ harmoninen keskiarvo
V[(⍋V)[⌈0.5×⍴V]]                        ⍝ mediaani (keskeisarvo)
=⌿0=400 100 4∘.|K                       ⍝ ovatko vuodet K (vvvv) karkausvuosia?
⍉0 100 100⊤K                            ⍝ päiväyksen VVVVKKPP pilkonta osiin VVVV KK PP
K×○÷180                                 ⍝ kulman muunto asteista radiaaneiksi
K×180÷○1                                ⍝ kulman muunto radiaaneista asteiksi
!X-1                                    ⍝ gammafunktio Γ(X)
÷Y×(X-1)!Y+X-1                          ⍝ beetafunktio Β(X,Y)
(2=+⌿0=(⍳S)∘.|⍳S)/⍳S                    ⍝ alkuluvut lukuun S asti
-\⍳S                                    ⍝ vuorotteleva sarja 1 ¯1 2 ¯2 3 ¯3..
W[1++\(⍳+/V)∊1++\V]                     ⍝ vektori (V[1]⍴W[1]),(V[2]⍴W[2]),..
A⊖((A←⍳S)-⌈S÷2)⌽(S,S)⍴⍳S×S              ⍝ S×S-taikaneliö (S pariton ja >2)
→⍙⌈⍳B                                   ⍝ loogisesta ehdosta B riippuva hyppy osoitteeseen ⍙
→0⌊⍳B                                   ⍝ loogisesta ehdosta B riippuva hyppy pois ohjelmasta
⍎B/'LAUSEKE'                            ⍝ loogisesta ehdosta B riippuva lausekkeen suoritus
(V,W)[⍋⍋B]                              ⍝ vektoreiden lomitus bittivektorin B ohjaamana
(⍴V)≥⍋(⍳⍴V),I                           ⍝ lavennusvektori, jossa nollia indeksien I jälkeen
1/K                                     ⍝ skalaari yksialkioiseksi vektoriksi, muut eivät muutu
(¯2↑1 1,⍴M)⍴M                           ⍝ skalaarin/vektorin muunto matriisiksi
((×/¯1↓⍴K),¯1↑1,⍴K)⍴K                   ⍝ sääntiön muunto matriisiksi
((V⍳V)=⍳⍴V)/V                           ⍝ toisintoalkioiden poisto vektorista
(V∊K)\(V∊K)/V                           ⍝ vektorista halutut alkiot nolliksi/väleiksi
(-+/∧\⌽M=' ')⌽M                         ⍝ merkkisääntiön tasaus oikealle
(+/∧\M=' ')⌽M                           ⍝ merkkisääntiön tasaus vasempaan
(-⌊0.5×+/∧\⌽M=' ')⌽M                    ⍝ merkkisääntiön keskitys
(V≠' ')/V                               ⍝ välilyöntien poisto vektorista
V>¯1↓0,V                                ⍝ binäärivekorin ykkösryhmien alkuykköset
V>1↓V,0                                 ⍝ binäärivektorin ykkösryhmien loppuykköset
(1-(V=' ')⊥1)↓V                         ⍝ vektorin loppuvälilyöntien poisto
(∨\V≠' ')/V                             ⍝ vektorin alkuvälilyöntien poisto
((⌽∨\⌽A)∧∨/A←V≠' ')/V                   ⍝ vektorin päissä olevien välien poisto
(A∨1⌽A←V≠' ')/V                         ⍝ vektorin perättäisten välien tiivistys
1+(⍴V)-(V≠' ')⊥1                        ⍝ vektorin viimeisen välilyönnin indeksi
(¯1⌽1↓(∨/M≠¯1⊖M),1)⌿M                   ⍝ järjestetyn matriisin uniikkirivit
(~A∊1,⍴V)/A←A/⍳⍴A←(1↓A,0)<A←~V∊'aeiouyäö'     ⍝ suomenkielisen sanan tavutuskohdat
```

2. APL2

APL:n laajennusperiaatteet

APL:n tultua yleiseen käyttöön 70-luvulla sen kehitys ei suinkaan jäänyt paikoilleen. Innostuneen käyttäjäkunnan kehitysehdotukset, erityisesti vuosittaisissa **APL-konferensseissa** esille tuodut, saivat eri tuotevalmistajat lisäilemään uusia ominaisuuksia omiin tulkkeihinsa. Muun muassa epäsäännöllisen datan (esimerkiksi henkilötietojen) käsittely oli tehtävä usein mutkikkaiden vippaskonstien avulla; niinpä jo varhain alettiin hahmotella ns. toisen sukupolven APL:n ominaisuuksia.

Eri tuotevalmistajat tekivät omia laajennuksiaan niin, että yhtenäisen kielen asemesta julkistettiin joukko enemmän tai vähemmän yhteensopivia toisen sukupolven APL-murteita. IBM:n julkistettua 1982 oman **APL2**:ksi nimetyn kielensä on kehitys onneksi vienyt siihen, että nykyään lähes kaikki aktiiviset valmistajat pyrkivät tekemään tuotteistaan **kielitasolla** mahdollisimman APL2-yhteensopivia.

APL2 on ainoa toisen sukupolven APL, jota tässä osassa käsitellään.

APL:n perusominaisuudet voidaan kiteyttää seuraavasti:

- dataa (tietoa) käsitellään sääntiöinä
- APL-sääntiö on suorakulmaisesti järjestettyjen samantyyppisten alkioiden joukko
- datan esitys moniulotteisena sääntiönä säätelee myös itse laskennan suoritusta
- kaikki luvut ovat reaalilukuja, käyttäjän ei tarvitse operoida eri esitystapojen välillä
- funktiot käsittelevät argumentteinaan sääntiöitä, niiden tuloksena on uusia sääntiöitä
- funktioiden suoritusjärjestys riippuu vain niiden sijainnista suoritettavassa lausekkeessa
- operaattorit käsittelevät operandeinaan skalaarifunktioita ja tuottavat näistä uusia funktioita
- notaatiosyntaksi on **yksinkertainen** (ei monimutkaisia hierarkioita tms.).

APL2:n tavoitteena oli laajentaa perus-APL:stä (”APL1:stä”) kieli, joka olisi tehokas, luotettava ja muodollisesti eheä. Sen tärkeimpiä ominaisuuksia ovat:

- turhien rajoitusten poisto tai olennainen lievennys
- APL-ilmaisujen jako **syntaksiluokkiin**
- lausekkeiden suoritusjärjestys määritellään **sidoshierarkian** avulla
- kaikki luvut ovat **kompleksilukuja**
- uusi sääntiöominaisuus, **sisäkkäisyys**, sallii epäsäännöllisen datan käsittelyn
- sääntiö on **tyyppirajoitukseton**
- jokainen (argumentillinen) funktio on **ambivalentti**
- operaattoreiden operandeina voi olla mikä tahansa ei-niladinen funktio
- omien operaattorien luonti
- useiden funktioiden toimintoja on laajennettu
- virhetilanteiden ohjelmallinen hallinta
- muutama uusi funktio ja operaattori.

APL2:n toteutuksessa oli kyse tärkeiksi katsottujen tekijöiden välisestä tasapainosta. Suunnittelukriteereistä tärkeimpinä pidettiin mahdollisimman suurta **APL1-yhteensopivuutta** sekä yksinkertaisuus-, säännönmukaisuus- ja käytettävyysnäkökohtia. Kielen uusien ominaisuuksien kehittelyssä tukeuduttiin sopiviin **identiteetteihin** (mm: `⍴M[I;J] ←→ (⍴I),⍴J` ja `⍴Y∘.αX ←→ (⍴Y),⍴X`).

Seuraavissa kappaleissa käydään pääpiirteissään läpi keskeisiä APL2-kielen kehityssuuntaviivoja ja käsitteitä. Tiettyä teoreettista otetta ei voi välttää, mutta satunnainen lueskelija voinee pikaselauksen jälkeen siirtyä suoraan funktioiden ja operaattoreiden kuvausosaan. Teoriaosuuden tekstien pääosa perustuu APL2:n kehitysryhmän johtajan James A. *Brownin* teksteihin, erityisesti ***The Principles of APL2*** -julkaisuun.

Syntaksiluokat

APL2-kielen elementit jaetaan kuuteen syntaksiluokkaan:

- sääntiöt (**data**, muuttujat, subjektit)
- monadiset ja dyadiset funktiot (**ohjelmat**, verbit)
- monadiset operaattorit (adverbit)
- dyadiset operaattorit (adverbit)
- sijoitusnuoli (kopula)
- hakasulkeet (kameleonttioperaatio).

HUOM: Niladinen (argumentiton) arvon palauttava funktio on tulkittavissa **vakioksi**, ja näin ollen se kuuluu sääntiöiden kanssa samaan syntaksiluokkaan.

APL2:n lausekeluokat ja sidoshierarkia

APL2-kielessä käytetään vain APL2-lausekkeita.
APL2-lauseke koostuu yhdestä tai useammasta APL2-ilmaisusta (alilausekkeesta).
APL2-ilmaisu on aina joko sääntiö-, funktio- tahi operaattori-ilmaisu.

sääntiöilmaisut

APL2-sääntiöilmaisu on aina **yksirivinen** vektori-ilmaisu. Vektori muodostetaan kirjoittamalla vektorin skalaarialkiot vierekkäin välilyönneillä eroteltuina. APL2 sallii **sekatyyppisen** datan (numeerisen ja merkkidatan) käsittelyn samassa vektorissa.

```
1 2 3 4 5                  ⍝ numeerinen vektori, pituus = 5
'A' 'B' 'C'                ⍝ merkkivektori, pituus = 3
2 'B' 5 7.1                ⍝ sekavektori, pituus = 4
```

Vierekkäisillä välilyönnein erotetuilla sääntiöillä (tavallisesti skalaareilla) on keskenään **vektorisidos**.
Esimerkiksi `I J` on vektori, jonka muodostavat alkiot `I` ja `J` niiden omasta sisäisestä rakenteesta riippumatta.

funktioilmaisut

Kaikki APL2-funktiot ovat **ambivalentteja**, sekä monadisia että dyadisia, ja niiden käyttötapa riippuu vain funktiokutsusta. Funktion argumenteilla on funktioonsa oikean ja vasemman argumentin **sidos**.
Funktioilla **ei** ole keskinäistä hierarkiaa, suoritusjärjestys riippuu vain funktioiden sijainnista lausekkeessa.
Suoritusjärjestystä voi tarvittaessa säätää **sulkein**.

Symboleja `∘` ja `→` käsitellään syntaktisesti funktioina.

Funktiolausekkeen `2×3+4` tulkinnassa on kaksi eri vaihtoehtoa: kolmonen on ensisijaisesti joko kertolaskun oikea argumentti: `(2×3)+4` tai yhteenlaskun vasen argumentti: `2×(3+4)`.

APL1:ssä funktiot suoritetaan oikealta vasempaan, joten jälkimmäinen tulkintatapa asetetaan etusijalle.
Vasemman argumentin sidos on oikean argumentin sidosta vahvempi.

Vektori-ilmaisun sisältävä lauseke `2+3  4×5` voidaan tulkita joko niin, että funktiosidos on vektorisidosta vahvempi: `(2+3)(4×5)` taikka päinvastoin: `2+(3  4×5)`.

Jälkimmäinen tulkinta, jossa vektori käsitellään yhtenä kokonaisuutena, on selkeästi APL1:n mukainen.
Vektorisidos on funktion argumenttisidoksia vahvempi.

operaattori-ilmaisut

Operaattoreilla johdetaan funktio- ja dataoperandeista uusia funktioita. Operaattorit voivat olla **operandiensa** suhteen joko monadisia tai dyadisia. Toisin kuin funktiot, operaattorit **eivät** voi olla ambivalentteja, ja siksi ne jaetaan **kahteen** eri syntaksiluokkaan: monadisiin ja dyadisiin operaattoreihin.

Operaattorilla on aina vasen operandisidos; dyadisella operaattorilla on lisäksi myös oikea operandisidos.

APL1:n operaattorien funktio-operandit on rajoitettu primitiiviskalaarifunktioihin. APL2:n operaattorit voivat sen sijaan saada operandikseen **minkä tahansa** (ambivalentin) funktion, myös itse tehdyn tai operaattorilla johdetun. Operandina voi yhtä hyvin olla myös sääntiö.

Kahden operaattorin yhdistetty lauseke `+.×/` voidaan tulkita joko `(+.×)/`, sisätulon reduktioksi, tai `+.(×/)`, yhteenlaskun ja tuloreduktion sisätuloksi.

Käytännöllisyys ja se, että operaattorit käyttäytyvät funktioihin nähden peilikuvamaisesti, asettavat APL2:ssa edellisen vaihtoehdon etusijalle. **Oikean operandin sidos on vasemman operandin sidosta vahvempi**.

Kaksioperandinen kaksiargumenttinen lauseke `A×.-B` voidaan tulkita joko siten, että sisätulo-operaattorin operandeina ovat kerto- ja vähennyslasku: `(A)×.-(B)` tahi sitten niin, että operandeina ovat kertolasku ja oikean argumentin vastaluku: `A×.(-B)`.

APL1:n mukaan sisätulon argumentit ovat `A` ja `B`, joten **oikea operandisidos on vasenta argumenttisidosta vahvempi**.

Koska sama symboli (ilmaisu) ei voi olla samanaikaisesti sekä operaattori että funktio, ei vasemman operandisidoksen ja vasemman argumenttisidoksen ristiriitaa voi esiintyä, ja siksi niiden välisellä vahvuuserolla ei ole käytännön merkitystä. Symmetrian vuoksi asetetaan kuitenkin **vasen operandisidos vahvemmaksi kuin vasen argumenttisidos.**

Jos operaattorit olisivat ambivalentteja, reduktio saisi lausekkeessa `+/A×B` oikeaksi operandikseen `A`:n, koska funktion argumenttisidos on operandisidosta heikompi, ja johdettu funktio `+/A` olisi kertolaskun vasen argumentti eli lauseke tulkittaisiinkin muodossa `(+/A)×B`. Tämä on ristiriidassa APL1:n käytännön ja yksinkertaisuusperiaatteen kanssa, joten operaattori **ei** voi olla ambivalentti ja näin ollen operaattorit on jaettava kahteen **eri** syntaksiluokkaan (reduktio on siten **monadinen** operaattori).

Operaattorilla **johdettu** funktio sen sijaan on ambivalentti, joten muodot `+/B` ja `A+/B` ovat yksikäsitteisiä ja syntaktisesti mahdollisia.

Lavennuslausekkeessa `1 0 1/A` on APL1:ssä vasempana operandina vektori `1 0 1` eikä skalaari `1`.
Vektorisidos on vasemman operandin sidosta vahvempi.

Oikean operandisidoksen vahvuudesta ei APL1 anna osviittaa, joten APL2:n kehittäjät valitsivat (lähinnä Sharp APL:n yhteensopivuutta ajatellen) **oikean operandisidoksen vektorisidosta vahvemmaksi**.

Esimerkiksi dyadiselle operaattorille `DOP` tulkitaan lauseke `+ DOP A B` muotoon `(+ DOP A)B`.
[Jälkikäteen J.A. Brown onkin myöntänyt tämän kohdan jääneen APL2:n kauneusvirheeksi!]

hakasulkeet

Hakasulkeita, joilla on vain **vasen** sidos, käytetään sekä sääntiöiden indeksointiin että funktioiden ja operaattoreiden suoritussuuntien määritykseen. Hakasulkeet ovat **kameleonttioperaatioita**; niillä aikaansaatu lauseke on syntaksiluokaltaan sama kuin lausekkeen ilman hakasuljeilmaisua.

Dyadisen operaattorin `DOP` sisältävä lauseke `+ DOP ⌽[1]` voidaan tulkita kahdella tavoin: joko `(+ DOP ⌽)[1]` tai `+ DOP(⌽[1])`. Jälkimmäinen tulkinta on selkeämpi ja käytännössä yksinkertaisempi, joten **hakasuljesidos on oikean operandin sidosta vahvempi**.

Hakasuljesidos on siten vektorisidostakin vahvempi, ja siksi lauseke `A B C[2]` **tulee** tulkita kuten `A B(C[2])`. Vektori-indeksointilausekkeet on siten tarvittaessa ympäröitävä sopivasti sulkeilla.

Esimerkiksi APL1:n lauseke `1 2 3[2]` on APL2:ssa aina kirjoitettava muotoon `(1 2 3)[2]`.
Tämä on **ainoa** merkittävä syntaksiero APL1:stä APL2:een siirryttäessä.

Tekstivektorin indeksointisyntaksi säilyy ennallaan, koska heittomerkit rajaavat sen yksiselitteisesti.

sijoitusnuoli

Sijoitusnuolella on sekä vasen sidos (sijoitussidos) että oikea sidos (datasidos).

Lausekkeessa `A←2+3` tulee `A`:n arvoksi `5`, joten **argumentin vasen sidos on vahvempi kuin sijoitusnuolen oikea sidos**.

Lausekkeessa `2+A←3` sijoitus `A`:han tehdään ennen yhteenlaskua. **Sijoitusnuolen vasen sidos on oikeaa argumenttisidosta vahvempi**.

Dyadisen operaattorin `DOP` sisältävä lauseke `+ DOP A←3` voidaan tulkita joko `(+ DOP A)←3` taikka `+ DOP(A←3)`. Jälkimmäinen tulkinta on selkeämpi, joten **sijoitusnuolen vasen sidos on oikean operandin sidosta vahvempi**.

APL2:n lopulliseksi sidoshierarkiaksi saadaan (vahvimmasta heikoimpaan):

```
hakasulkeet
 sijoitusnuoli vasempaan
  oikea operandi
   vektori
    vasen operandi
     vasen argumentti
      oikea argumentti
       sijoitusnuoli oikealle.
```

HUOM:
- **oikeanpuoleisin** funktio, jonka argumentit ovat valmiit, suoritetaan ensin
- APL2-lausekkeen arvo tulostetaan ruudulle, ellei sen viimeinen suoritettava syntaksitoimenpide ole sijoitus (tavallinen tai valintasijoitus).

Vektoriesitys

Vektoriesityksessä (*vector notation, strand notation*) voidaan vektoriin kuuluvat yksittäiset paljaat skalaarit syöttää yksitellen joko välilyöntien tai sulkeiden erottamina. Erotinvälilyöntien lukumäärä ei muuta lausekkeen arvoa. **Yksittäisten** merkkien syötössä ovat välilyöntierottimet kuitenkin olennaisia.

```
        1 2 'A'  3  'B'   'C'
1 2 A 3 BC
        'A' 'P' 'L' '2'
APL2
        'A''P''L''2'
A'P'L'2
```

Sulkeita käytetään sekä ohjaamaan lausekkeiden suoritusjärjestystä että ilmaisemaan sääntiöryhmittelyä. Sulkeet ovat tarpeettomat, jos ne eivät samanaikaisesti **sekä** erota **ja** ryhmitä.

```
        (2 3) ←→ 2 3                ⍝ eivät erota
        (2)(3) ←→ 2 3               ⍝ eivät ryhmitä
        (2 3)(4 5)                  ⍝ sulkeet ovat tarpeen
```

Sulkeita voi haluttaessa käyttää selventämään sääntiö-, funktio- tai operaattori-ilmaisua.

```
        (1 2)+.÷(2 3)               ⍝ argumentit
        1 2(+).(÷)2 3               ⍝ operandit
        1 2+(.)÷2 3                 ⍝ operaattori
        1 2(+.÷)2 3                 ⍝ johdettu funktio
        1 2+.÷2 3                   ⍝ sama sulkeitta
```

Sulkeilla erotetun lausekkeen voi **aina** korvata vastaavalla APL-ilmaisulla.

```
        A←1 2
        (1 2)3(4 5) ←→ A 3(4 5)
```

Pelkistä paljaista merkkiskalaareista koostuva vektori voidaan sulkea yksiin heittomerkkeihin.

```
        'A' 'B' 'C' ←→ 'ABC'
```

Sulkeissa oleva lauseke suoritetaan **ennen** kuin sitä käytetään argumenttina tahi sääntiölausekkeessa.

Ohjelmoitavat operaattorit

Käyttäjän ohjelmoimat operaattorit eroavat tavallisista funktioista vain otsikkoriviltään: operaattorin nimi ja operandit erotetaan **sulkeilla**.

Monadisen (`MOP`) ja dyadisen operaattorin (`DOP`) otsikkorivien muodot ovat:

```
[tulos←] [v_arg](v_oper MOP )      [o_arg]{;lokaalimuuttuja}
[tulos←] [v_arg](v_oper DOP o_oper)[o_arg]{;lokaalimuuttuja}
```

Operaattoria suoritettaessa operandien nimet korvautuvat vastaavilla **funktioilla** tai **sääntiöillä**. Esimerkiksi matriisien `M1` ja `M2` kertolasku operaattorilla `STULO` sekä funktioilla `plus` ja `kertaa` sujuu lausekkeella:

```
        M1 plus STULO kertaa M2     ⍝ ←→ M1 +.× M2
```

Korvaussäännöt

APL2-lausekkeessa voidaan alilauseke korvata vastaavan arvon tuottavalla ilmaisulla **lausekkeen arvon** muuttumatta.

```
(1↓⍳4)-1  ←→  (1+⍳3)-1                        ⍝ sisältövastaavuus
```

Sulkeissa oleva lauseke voidaan korvata vastaavalla sääntiöilmaisulla **lausekkeen arvon** muuttumatta.

```
A←2 3 4
2 3 4,5 ←→ (2 3 4),5 ←→ (A),5 ←→ A,5
```

Sääntiön alkio voidaan korvata toisella vastaavan alkion tuottavalla ilmaisulla **sääntiön arvon** muuttumatta.

```
2 3 4 ←→ 2 (3) 4 ←→ 2(4-1)4
```

Alilauseke voidaan korvata samaan syntaksiluokkaan kuuluvalla ilmaisulla **lausekkeen syntaksin** muuttumatta.

```
A+B ←→ A(+)B <~> A(*)B ←→ A*B           ⍝ syntaksivastaavuus
A++/B ←→ (A)(+)(+/)(B) <~> (C)(*)(∧⍀)(D) ←→ C*∧⍀D
```

Kompleksiluvut

Jokainen APL2:n luku on kompleksiluku. **Aito** kompleksiluku esitetään aina muodossa `xJy`, missä `x` vastaa reaaliosaa ja `y` imaginaariosaa; reaalilukujen imaginaariosaa **ei** tulosteta. Kompleksiluku voidaan syöttää myös kulmamuodoissa (polaarimuodoissa) `rDα` taikka `rRα`, missä `r` vastaa etäisyyttä origosta ja α kulma-arvoa asteina (*D*) tai radiaaneina (*R*).

Seuraavien skalaarifunktioiden arvoaluetta ja määritelmiä on laajennettu kompleksilukualueelle (kompleksiluvun `Z` reaali- ja imaginaariosat ovat `X` ja `Y`, eli `Z ←→ X+Y×0J1`):

- monadinen + on kompleksiluvun **liittoluku** *(conjugate)* (`+Z ←→ X+(-Y)×0J1`)
- monadinen | on kompleksiluvun **etäisyys** *(magnitude)* origosta eli kompleksiluvun esittämän tasopisteen etäisyys origosta (`|Z ←→ (+/X Y*2)*0.5 ←→ (Z×+Z)*0.5`)
- monadinen × on kompleksiluvun **suuntafunktio** *(direction)*.

Trigonometristen funktioiden joukkoa on vielä laajennettu kompleksilukuoperaatioilla:

`8○Z`	$\sqrt{-1-z^2}$	`¯8○Z`	$-\sqrt{-1-z^2}$
`9○Z`	`Z`:n reaaliosa (`X`)	`¯9○Z`	`Z`
`10○Z`	`|Z` (etäisyys origosta)	`¯10○Z`	`+Z` (liittoluku)
`11○Z`	`Z`:n imaginaariosa (`Y`)	`¯11○Z`	`Z×0J1`
`12○Z`	`Z`:n vaihekulma (φ(`Z`))	`¯12○Z`	`*Z×0J1`.

Kompleksiluvulle `Z` pätevät seuraavat identiteetit:

```
Z ←→ ¯10 ¯11 +.○  9 11 ∘.○ Z ja
Z ←→  ¯9 ¯12 ×.○ 10 12 ∘.○ Z.
```

Sisäkkäiset sääntiöt

APL2:n sääntiön alkiona voi olla toinen sääntiö. Jos jokainen sääntiön alkio on **paljas** (ei-sisäkkäinen) skalaari, on sääntiökin paljas. Korostettaessa APL1:n sääntiöiden **alkioiden** *(element)* ja APL2:n sääntiöiden alkioiden rakenne-eroa, käytetään toisinaan APL2-sääntiöiden alkioista myös nimitystä **solu** *(item)*.

Sisäkkäisyyden mitta on **syvyys** (kerroksisuus). Paljaan skalaarin syvyys on = `0`, paljaan sääntiön syvyys on = `1` ja sisäkkäisen sääntiön syvyys on = `1` + **sisäkkäisimmän** alkion syvyys. Sääntiön sisäkkäisyyden kertoo monadinen syvyysfunktio `≡`. Ruudulle tulostettaessa alkioiden väliin sijoitetaan sisäkkäisyydestä riippuva lukumäärä välilyöntejä ja tulostetta myös sisennetään syvyyttä vastaavasti.

```
        ≡(1 2)(3(4 5))
3
        (1 2)(3(4 5))
 1 2  3  4 5                      ⍝ sisennys, välilyönnit!
```

Sääntiön voi muuttaa sisäkkäiseksi skalaariksi monadisella <u>kätkentä</u>funktiolla `⊂`, ja sisäkkäisyyden voi poistaa monadisella <u>paljastus</u>funktiolla `⊃`. **Paljasta** skalaaria ei kuitenkaan voi kätkeä (ns. kelluva sisäkkäisyys).

```
        313               ←→ ⊂313             ←→ ⊂⊂⊂313
        (1 2)(3(4 5))     ←→ (⊂1 2),⊂3(4 5) ←→ (⊂1 2),⊂3,⊂4 5
        A B C             ←→ (⊂A),(⊂B),(⊂C)
```

Hakasuljeindeksointi ei vaikuta sisäkkäisen sääntiön syvyyteen, sillä voidaan valita vain uloimman kerroksen (kuoren) alkioita. Sen sijaan dyadisella <u>poiminta</u>funktiolla `⊃` voi valita ja **paljastaa** sääntiöstä **minkä tahansa** alkion. Esimerkiksi sisäkkäiselle ei-tyhjälle vektorille `1⊃A ←→ ⊃A[1]`.

Sääntiön sisäkkäisyyttä voi tutkia tuotteen mukana tulevalla *`DISPLAY`*-funktiolla, joka antaa tietoa sääntiön sisäkkäisyyden lisäksi myös sen muusta rakenteesta ja sisällöstä.

```
        DISPLAY 1 1 2⍴(1 4⍴('APL',2)'=' 'FUN'(○1))((⍳0)(0 1⍴'A'))
┌┌→─────────────────────────────────────────────────────────────┐
↓↓ ┌→────────────────────────────────────────┐ ┌→──────────────┐ │
││ ↓ ┌→─────┐   ┌→──┐                       │ │ ┌⊖┐ ┌→┐       │ │
││ │ │APL 2│ = │FUN│ 3.141592654           │ │ │0│ ⌽ │       │ │
││ │ └+─────┘ ─ └───┘                       │ │ └~┘ └─┘       │ │
││ └∊────────────────────────────────────────┘ └∊──────────────┘ │
└└∊─────────────────────────────────────────────────────────────┘
```

Alkion sisäkkäisyys = sitä ympäröivien rajojen lukumäärä. Paljaalla skalaarilla ei ole rajoja, paljaan merkkiskalaarin alle tulostetaan yksi alaviiva. Jokaisen laatikon vasemman yläkulman symbolit kertovat, onko kyseessä vektori (→), matriisi (→ ja ↓) vai tyhjä vektori (⊖) tahi matriisi (⌽). Vasemman alakulman merkintä ilmaisee, onko alkion sisältö tyypiltään sisäkkäinen (∊), numeerinen (~), merkki- vai sekamuotoinen (+). Vasemman reunan pystyviivojen lukumäärä näyttää vielä alkion moniulotteisuuden.

Myös tyhjät sääntiöt voivat olla sisäkkäisiä (eli niilläkin voi olla oma rakenne).

```
        DISPLAY 0 2⍴⊂22 33
┌→────────────────┐
⌽ ┌→───┐ ┌→───┐ │
│ │0 0│ │0 0│ │
│ └~───┘ └~───┘ │
└∊────────────────┘
```

Valintasijoitus

Sijoitusnuolen vahva vasen sidos mahdollistaa ilmaisut, joissa sijoitettavana on sääntiön nimen asemesta olemassa olevaan sääntiöön tai sen osaan kohdistuva valintalauseke (valintasijoitus).

```
        A[1]←3              ⍝ alkioon sijoitus indeksoimalla
        (1↑A)←3             ⍝ sama valintasijoituksella
        (1⊃A)←3 4           ⍝ alkion arvon vaihto: A[1] ←→ ⊂3 4
        (2↓2 3,A)←3 4       ⍝ sääntiön sisällön korvaus: A ←→ 3 4
```

Jos on mahdollista kirjoittaa lauseke, jolla poimitaan sääntiöstä alkioita, sen voi panna sulkeisiin ja sijoituksella muuttaa valittujen alkioiden sisältöä. Sääntiötä, jonka alkioita näin käsitellään, kutsutaan **kohdesääntiöksi**.

HUOM: Valintalausekkeessa sallitut valintafunktiot sekä -operaattorit riippuvat käytettävästä APL2-toteutuksesta.

Poimintafunktiolla ⊃ voi valita vain yhden sijoitusalkion. Valintalausekkeessa voi lisäksi käyttää hakasulje-indeksointia ja (muuhun kuin valintaan) mitä tahansa funktioita ja operaattoreita.

Monen sääntiön valintasijoituksesta *(selective specification, selective assignment)* havaitaan, että **monisijoitus** *(multiple assignment)* on valintasijoituksen erikoistapaus: `((⊂A),(⊂B),(⊂C))←X Y Z ←→ (A B C)←X Y Z.`

Monisijoituksesta nähdään taas, että tavallinen (yhteen muuttujaan) sijoitus on monisijoituksen erikoistapaus:
`(A B C)←X Y Z => (¯2↓A B C)←¯2↓X Y Z ←→ (A)←X ←→ A←X.`

Monen sääntiön samanaikaista valintasijoitusta **ei** ole yleisesti määritelty.

Valintasijoituksessa käydään läpi kolme eri vaihetta:

- **Valitaan** kohdesääntiöstä kiinnostavat alkiot valintalausekkeella.
- **Korvataan** valitut alkiot. Jos korvataan vain yksi alkio, korvataan koko kohdesääntiön alkio sijoitettavalla datalla. Muussa tapauksessa data viedään kohdesääntiön korvattaviin alkioihin data-alkio-pareittain (mahdollisin skalaarilaajennuksin).
- **Palautetaan** lausekkeen arvo = sijoitusnuolen oikeanpuoleinen arvo ennen sijoitusta.

```
        A←1 2 3 1 5 2 4 ⋄ 1+ ((A=3)/A)←99
100
        A
1 2 99 1 5 2 4
        ((1=2|A)/A)←'α' ⋄ A
α 2 ααα 2 4                     ⍝ ←→ 'α' 2 'α' 'α' 'α' 2 4
```

Poimintafunktiolla voi valita jopa paljaasta skalaarista, joka tällöin mielletään riittävästi kätketyksi.

```
        (((⍳0)(⍳0))⊃A)←3 4      ⍝ vektori "toiseen kerrokseen"
        DISPLAY A               ⍝ ≡A ←→ 3
┌─────────────┐
│ ┌─────────┐ │
│ │ ┌→───┐  │ │
│ │ │3 4 │  │ │
│ │ └~───┘  │ │
│ └∊────────┘ │
└∊────────────┘
        (((⍳0)(⍳0))⊃A)←7 ⋄ A    ⍝ palautus skalaariksi
7
```

Paljaan skalaarin kätkentä

Eri APL1-laajennusten suurin periaatteellinen ero on siinä, onko kätketty paljas skalaari sisäkkäinen (**ankkuroitu sisäkkäisyys**: Sharp APL) vai ei (**kelluva sisäkkäisyys**: APL2, APL*PLUS, Dyalog APL, APLX).

Seuraavassa ovat pääkohdat J.A. Brownin esityksestä, jossa hän osoitti miksi APL2:n kelluva sisäkkäisyys tuntuu ankkuroitua luontevammalta.

Deduktiivinen tapa tutkia ongelmaa on lähteä liikkeelle yleisestä tapauksesta, tehdä tästä sopivia havaintoja ja lopuksi laajentaa tehdyt päätelmät erikoistapauksiin kuten paljaisiin skalaareihin, jos mahdollista. Lähdetään liikkeelle **tasarakenteisista** sisäkkäisistä sääntiöistä, joiden alkiot ovat keskenään samanmuotoisia ja -syvyisiä. Tämä on tarpeeksi yleinen lähtötilanne: jos joku yleinen sääntö pätee tasarakenteiselle sääntiölle, on sen oltava voimassa myös muunlaisille sääntiöille.

Sääntiön täydelliseen määrittelyyn tarvitaan vain kaksi tekijää: **sisältö** ja **rakenne**. Sääntiön sisällön urkintaan on olemassa perusfunktio, monadinen `∊`, joka palauttaa argumenttisääntiön sisällön paljasrakenteisena vektorina.

Sääntiön rakenteelle ei ole olemassa vastaavaa primitiivifunktiota, joten määritellään tasarakenteiselle sääntiölle hypoteettinen monadinen rakennefunktio `STRUCT` seuraavasti: funktion tuloksena on argumenttinsa ääretön **rakennevektori**, jonka ensimmäinen alkio on argumenttisääntiön kokovektori, seuraava alkio on argumentin ensimmäisen alkion kokovektori, seuraava alkio on argumentin ensimmäisen alkion ensimmäisen alkion kokovektori, ja niin edespäin. Sääntiön ensimmäisen alkion poimintaan voidaan käyttää monadista **eka**funktiota `↑` `(↑K ←→ ⎕IO⊃,K)`.

Esimerkiksi jos `A←3 2⍴⊂5⍴⊂4 3⍴0`, on

```
STRUCT A ←→ (3 2) (,5) (4 3)   (⍳0)    (⍳0)..
         ←→  (⍴A)(⍴↑A)(⍴↑↑A)(⍴↑↑↑A)(⍴↑↑↑↑A)..
```

Rakennevektorin sisällön tulo = tasarakenteisen sääntiön paljaiden skalaarien lukumäärä.

```
      ×/∊ STRUCT A                         ⍝ ←→ ⍴∊A
360
```

Jos sääntiö `A` kätketään, siitä tulee sisäkkäinen skalaari, eli

```
STRUCT ⊂A ←→ (⍳0)(3 2)(,5)(4 3)(⍳0)(⍳0)(⍳0)..
      ∊⊂A ←→ ∊A                    ⍝ sisältö säilyy samana
```

Rakennevektorin **eteen** tulee tyhjä vektori, siispä sisältö ei muutu, ainoastaan syvyysrakenne!

Paljaan skalaarin ja kätketyn skalaarin rakennevektoreiksi saadaan nyt:

```
STRUCT  5 ←→     (⍳0)(⍳0)(⍳0)..        ⍝ ←→ (⍴5)(⍴↑5)..
STRUCT ⊂5 ←→ (⍳0)(⍳0)(⍳0)(⍳0)..
```

Kätkeminen lisää siis rakennevektorin eteen yhden `⍳0`:n. Paljaan skalaarin rakennevektori on jo **ääretön** jono tyhjiä vektoreita, eikä yhden lisäys eteen sitä muuta. Niinpä paljaan skalaarin ja kätketyn paljaan skalaarin rakennevektorit ovat identtiset, eli **paljaan skalaarin rakenne ei kätkettäessä muutu**.

HUOM: Määritellään vielä toinen (äärellinen) rakennefunktio `STRUCT1`, jonka tuloksena on rakennevektori ilman lopussa olevia tyhjiä vektoreita. Tällöin on voimassa identiteetti: `≡X ←→ ↑⍴STRUCT1 X`.

Tyyppi ja prototyyppi

Sääntiön *X* **tyyppi** (tyyppisääntiö)(*type*) saadaan lausekkeella `↑0ρ⊂X`. Tyyppisääntiö on alkuperäisen sääntiön kanssa samanrakenteinen, mutta siinä on nolla jokaisen paljaan luvun ja välilyönti jokaisen paljaan merkin paikalla.

```
        ↑0ρ⊂(⍳3),'A',⍳3
0 0 0   0 0 0                   ⍝ ←→ 0 0 0 ' ' 0 0 0
```

Sääntiön *X* **prototyyppi** (*prototype*) saadaan lausekkeella `↑0ρ⊂↑X` , eli prototyyppi = **ensimmäisen** alkion tyyppi.

```
        ↑0ρ⊂↑(⍳3),'A',⍳3
0
```

Prototyyppiä käytetään tarvittaessa otto-, eka-, lavennus-, monistus- ja paljastusfunktioille sääntiöiden **täyte-alkiona** (*fill element*).

```
        ↑0ρ⊂↑A←(1 2)'ABC'
0 0                             ⍝ ←→ (0 0)
        4ρA
 1 2  ABC  1 2  ABC             ⍝ ←→ (1 2)'ABC'(1 2)'ABC'
        4↑A
 1 2  ABC  0 0  0 0             ⍝ ←→ (1 2)'ABC'(0 0)(0 0)

        DISPLAY 1 1 1 0 1\((2 3)(4(5 6)))7 8 9
┌→─────────────────────────────────────────────────────────┐
│ ┌→────────────────────┐       ┌→────────────────────┐     │
│ │ ┌→──┐ ┌→──────────┐ │ 7 8  │ ┌→──┐ ┌→──────────┐ │ 9   │
│ │ │2 3│ │   ┌→──┐   │ │      │ │0 0│ │   ┌→──┐   │ │     │
│ │ └~──┘ │ 4 │5 6│   │ │      │ └~──┘ │ 0 │0 0│   │ │     │
│ │       │   └~──┘   │ │      │       │   └~──┘   │ │     │
│ │       └∊──────────┘ │      │       └∊──────────┘ │     │
│ └∊────────────────────┘      └∊────────────────────┘     │
└∊─────────────────────────────────────────────────────────┘
```

Täytefunktio

Primitiivifunktioilla ja kukin- sekä sisätulo-operaattoreilla johdetuilla funktioilla, joilla on argumenttina tyhjä sääntiö, sovelletaan kyseisen funktion mukaista **täytefunktiota** *(fill function)* alkuperäisen funktion asemesta. Esimerkiksi skalaarifunktioiden täytefunktiona on `≠`.

Täytefunktion argumentteina on tyhjän sääntiön asemesta tämän prototyyppi; tyhjän paljaan tekstivektorin prototyyppi on tällöin = `0`. Sisäkkäisille sääntiöille täytefunktiota käytetään rekursiivisesti.

```
        DISPLAY ''÷''
┌⊖┐
│0│
└~┘

        DISPLAY ÷0 2ρ⊂1 2 3
┌→──────────────────┐
⌽ ┌→────┐ ┌→────┐   │
│ │0 0 0│ │0 0 0│   │
│ └~────┘ └~────┘   │
└∊──────────────────┘
```

Skalaarifunktioista

Monadisille skalaarifunktioille sovelletaan seuraavia käsittelysääntöjä (yhteensopivuusehtoja):
- paljaalle skalaariargumentille käytetään funktiota sellaisenaan
- ei-tyhjälle argumentille käytetään funktiota jokaiselle paljaalle alkiolle erikseen
- tyhjälle argumentille käytetään täytefunktiota, jonka argumenttina on sääntiön prototyyppi.

Dyadiselle skalaarifunktiolle sovelletaan taas seuraavia käsittelysääntöjä:
- jos molemmat argumentit ovat paljaita skalaareja, käytetään funktiota sellaisenaan
- tyhjälle argumentille käytetään täytefunktiota, jonka argumenttina on sääntiön prototyyppi
- samanmuotoisille argumenteille funktiota sovelletaan vastinalkiopareittain
- jos toinen argumentti on skalaari tai yksialkioinen vektori, sitä käytetään **jokaisen** toisen argumentin alkion kanssa (skalaarilaajennus).

Sisäkkäisille sääntiöille näitä sääntöjä käytetään rekursiivisesti.

```
         DISPLAY 2 (3 4)(5 6 7) + ((8 9) 10) 11 (12 13 14)
┌→───────────────────────────────────────────┐
│ ┌→─────────────────┐ ┌→──────┐ ┌→──────────┐ │
│ │ ┌→──────┐        │ │14 15│ │17 19 21│ │
│ │ │10 11 │ 12    │ └~──────┘ └~──────────┘ │
│ │ └~──────┘        │                         │
│ └∊─────────────────┘                         │
└∊───────────────────────────────────────────┘
```

Dyadisen skalaarifunktion suoritussuunnan voi määrittää suuntamerkinnällä (`K1 α[W] K2`).

```
         V←1 10 100 1000
         M←4 3⍴⍳12
         V×[1]M
    1     2     3
   40    50    60
  700   800   900
10000 11000 12000
```

Sama suunta **ei** saa esiintyä suuntavektorissa `W` kuin kerran.
Akseleille on voimassa: `⍴,W ←→ (⍴⍴K1)⌊⍴⍴K2` ja `∧/W∊⍳(⍴⍴K1)⌈⍴⍴K2`.

Argumentit ovat yhteensopivia, jos `⍴W ←→ (⍴K1)[W]` (tai `⍴W ←→ (⍴K2)[W]`), kun `W≡W[⍋W]`.

Tuloksen koko = moniulotteisemman argumentin koko.

```
         M+[1 3]100×3 2 4⍴⍳24
 101  202  303  404
 501  602  703  804

 905 1006 1107 1208
1305 1406 1507 1608

1709 1810 1911 2012
2109 2210 2311 2412
```

APL2-funktiot ja -operaattorit

Esitellään seuraavaksi uudet operaattorit ja funktiot sekä APL1-funktioista ne, joiden ominaisuuksia on APL2:ssa laajennettu. Käytetään seuraavia merkintöjä:

- *S* skalaari
- *V* vektori
- *W* suuntavektori (yksi tai useampia kokonaislukuja)
- *M* matriisi
- *K* kaikki sääntiöt
- *I* kokonaislukuindeksi
- α funktio.

APL2-operaattorit

kukin (jokaiselle) `α¨K      K1 α¨ K2`

Kukin-operaattori *(each)* on monadinen (yksioperandinen) operaattori.

Monadinen operandifunktio α kohdistetaan jokaiselle argumentin *K* alkiolle.
Argumentin syvempiin kerroksiin päästään operaattoria toistamalla.

Operaattorille on voimassa identiteetti: `I⊃α¨K  ←→  α(I⊃K)`.

```
        A←⍳¨⍳5    ⋄ A                  ⍝ ←→ (⍳1)(⍳2)(⍳3)(⍳4)(⍳5)
 1  1 2  1 2 3  1 2 3 4  1 2 3 4 5
        ⍴A
5
        ⍴¨'APL' (2⍴⊂'APL2')
 3  2                                  ⍝ ←→ (,3)(,2)
        ⍴¨¨'APL' (2⍴⊂'APL2')
          4  4                         ⍝ ←→ (3⍴⊂⍳0)((,4)(,4))
```

Dyadinen operandifunktio α kohdistetaan jokaiseen argumenttipariin. Skalaariargumentti laajennetaan toisen argumentin rakenteen mukaiseksi.

Operaattorille on voimassa identiteetti: `I⊃K1 α¨ K2  ←→  (I⊃K1)α(I⊃K2)`.

```
        2⍴¨3 4 5                       ⍝ ←→ 2 2 2⍴¨3 4 5
 3 3  4 4  5 5
        (⊂2 3)⍴¨3 4 5                  ⍝ skalaarilaajennus
 3 3 3  4 4 4  5 5 5                   ⍝ ←→ (2 3)(2 3)(2 3)⍴¨3 4 5
 3 3 3  4 4 4  5 5 5
        A←(10 20)(30 40)(50 60)
        +/A                            ⍝ ←→ (+/10 30 50)(+/20 40 60)
 90 120
        +/¨A                           ⍝ ←→ (+/10 20)(+/30 40)(+/50 60)
30 70 110
```

Jos operandifunktio on skalaarifunktio, ei kukin-operaattorilla ole vaikutusta funktion toimintaan.
Tyhjällä argumentilla käytetään sen prototyyppiä funktion **täytefunktion** argumenttina.

```
        1 2 3 +¨ 4 5 6      ⍝ ←→ 1 2 3 + 4 5 6
5 7 9
```

monistus (toisto), pelkiste, liukuva pelkiste `V/[S]K    α/[S]K    S1 α/[S2]K`

APL2:ssa etukenosymboli `/` on **aina** operaattori. Sen avulla johdetaan eri tyyppisiä funktioita:

monistusfunktio `V/[S]K` *(replicate)* johdetaan vektorioperandilla `V`.
Monistuksella toistetaan operandivektorin mukaisesti sääntiön `K` tasoja suunnassa `S`.

Jos `V` on skalaari tai yksialkioinen vektori (`V≥0`), sitä käytetään kaikille monistustasoille, muutoin tulee olla `(ρK)[S] ←→ +/V≥0`. Suuntamerkinnän puuttuessa on oletuksena viimeinen suunta (`V⌿K`:lle ensimmäinen).

Jos `V`:n toistinalkio on positiivinen tai nolla, suuntaa `S` vastaavan tason alkioita monistetaan toistinalkion verran. Negatiivisella toistinalkiolla monistetaan suunnan `S` **täytealkiota** toistinalkion itseisarvon verran.

```
         1 2 ¯1 3 ¯2 0/6 7 8 9
6 7 7 0 8 8 8 0 0
         0 1 ¯1 2⌿⊃'EI NÄY' 'YKSI' 'TUPLAA'
YKSI

TUPLAA
TUPLAA
```

pelkistefunktiossa `α/[S]K` *(reduce)* voi operandina olla mikä tahansa dyadinen funktio.
Pelkistys vähentää ei-skalaarisen argumentin ulotteisuutta yhdellä.

```
         ×/[1]3 4⍴⍳12                ⍝ ←→ ×⌿3 4⍴⍳12
45 120 231 384
         A←,/'A' 'BC' 'DEF' 'GHIJ'
         DISPLAY¨ A (⍴A) (≡A)        ⍝ vektorin pelkiste on skalaari
 ┌──────────────────┐ ┌⊖┐
 │ ┌→────────────┐  │ │0│ 2
 │ │ABCDEFGHIJ   │  │ └~┘
 │ └─────────────┘  │
 └∊─────────────────┘
```

liukuva pelkiste (askeltava reduktio) `S1 α/[S2]K` *(reduce n-wise)* on dyadinen funktio, jonka vasen argumentti `S1` määrittelee sen **ikkunan** (liukuma-alueen) leveyden, jolle funktiota pelkistetään suunnassa `S2`.

`S1` ja `S2` ovat kokonaislukuja, joko skalaareja tai yksialkioisia vektoreita. Jos `S1` on negatiivinen, ikkunan sisältö **heijastetaan** ennen pelkistystä. Tuloksen pituus pelkistesuunnassa on `1+(ρK)[S2]-|S1`.

```
         3+/⍳5                       ⍝ ←→ (+/1 2 3),(+/2 3 4),(+/3 4 5)
6 9 12
         (4+/1 4 9 16 25 36)÷4       ⍝ juokseva keskiarvo
7.5 13.5 21.5
         ¯2-/1 4 9 16 25 36          ⍝ viereisten alkioiden erotukset
3 5 7 9 11
         ¯2,/'ABCD'
BA CB DC
```

Jos `S1=0`, käytetään funktion **identtisyysalkiota**.

```
         0×/⍳5
1 1 1 1 1 1                          ⍝ huomaa pituus!
```

lavennus, selaus (kertymä, kumulointi) V\[S]K V\K α\[S]K α\K

APL2:ssa takakenosymboli \ on **aina** operaattori. Sen avulla johdetaan eri tyyppisiä funktioita:

lavennusfunktiolla V\[S]K *(expand)* lavennetaan sääntiötä K suunnassa S **loogisen** vektorin V avulla. Jos suuntaa ei anneta, oletuksena on viimeinen ulottuvuus (V⍀K:lle ensimmäinen). V:n nolla-alkio laajentaa tulossääntiötä laajennussuunnan **täytealkiolla**, ykkösalkiot osoittavat K:n alkuperäisen datan sijoittumiskohdat. Ohjainvektorille V on voimassa: +/V ←→ (⍴K)[S].

```
          1 0 1 0 1 0 1\'APL2'
A P L 2
          A←2 3 4⍴⍳24 ⋄ ((,A)[1 3 13 16])←'ACDE'
          A
 A  2  C  4
 5  6  7  8
 9 10 11 12

 D 14 15  E
17 18 19 20
21 22 23 24
          1 1 1 0 1\[3]A
 A  2  C    4
 5  6  7 0  8
 9 10 11 0 12

 D 14 15    E
17 18 19 0 20
21 22 23 0 24
```

selausfunktion α\[S]K *(scan)* tuloksena on K:n muotoinen sääntiö, jonka kukin alkio on dyadisen funktion α pelkiste sille K:n osasääntiölle, jossa suunnassa S on alkioita vain tähän alkioon asti. Jos suuntaa ei anneta, oletuksena on viimeinen suunta (α⍀K:lle ensimmäinen).

```
          +\(1 2)(3 4)(5 6)
 1 2  4 6  9 12
          +\¨(1 2)(3 4)(5 6)
 1 3  3 7  5 11
          DISPLAY ,\'A' 'BC' 'DEF' 'GHIJ'
┌→─────────────────────────────────────────────┐
│     ┌→──┐  ┌→─────┐  ┌→─────────┐ │
│  A  │ABC│  │ABCDEF│  │ABCDEFGHIJ│ │
│  ─  └───┘  └──────┘  └──────────┘ │
└∊─────────────────────────────────────────────┘
          +\[1]3 4⍴⍳12                      ⍝ ←→ +⍀3 4⍴⍳12
 1  2  3  4
 6  8 10 12
15 18 21 24
          ,\[1]3 3⍴⍳9                       ⍝ ←→ ,⍀3 3⍴⍳9
      1       2       3
    1 4     2 5     3 6
  1 4 7   2 5 8   3 6 9
```

APL2-funktiot

valinta (indeksifunktio) `V⌷[W]K` `V⌷K` {⎕IO}

Dyadinen valintafunktio *(index)* valitsee oikeasta argumenttisääntiöstä `K` vasemman indeksiargumentin `V` osoittamat alkiot kokonaislukuvektorin `W` määräämissä suunnissa. Jos suuntamerkintä puuttuu, oletuksena ovat **kaikki** suunnat: `V⌷K ←→ V⌷[⍳⍴⍴K]K`. Indeksivektorin pituus = suuntien lukumäärä (`⍴V ←→ ⍴W`), esimerkiksi kolmiulotteiselle sääntiölle `M`: `I J⌷[1 3]M ←→ M[I;;J]`.

Valintafunktion ja hakasuljeindeksoinnin välillä on identiteetti: `I J K⌷M ←→ M[I;J;K]`. Skalaarinkin valinta on mahdollista lausekkeella `(⍳0)⌷S`.

```
          A←11 12 13 14 ⋄ 3⌷A          ⍝ ←→ A[3]
13
          (⊂2 1)⌷A                     ⍝ ←→ A[2 1]
12 11
          M←2 2⍴11 12 13 14 ⋄ 2 1⌷M    ⍝ ←→ M[2;1]
13
          2(2 1)⌷M                     ⍝ ←→ M[2;2 1]
14 13
          2⌷[1]M                       ⍝ ←→ M[2;]
13 14
```

sisältö (listaus) `∊K`

Monadisen sisältöfunktion *(enlist)* tuloksena on argumentin **paljas** sisältövektori (rivisuunnassa).

```
          A←(2 2⍴⍳4)(2 2⍴(5 6(2 2⍴7 8 9 10)11))'ABCD'
          DISPLAY¨ A (∊A)
┌→─────────────────────────────────────────────┐   ┌→─────────────────────────────────┐
│ ┌→───┐  ┌→─────────────┐  ┌→───┐             │   │1 2 3 4 5 6 7 8 9 10 11 ABCD│
│ ↓1 2│  ↓  5        6   │  │ABCD│             │   └+─────────────────────────────────┘
│ │3 4│  │               │  └────┘             │
│ └~───┘  │ ┌→────┐       │                     │
│         │ ↓7   8│  11   │                     │
│         │ │9  10│       │                     │
│         │ └~────┘       │                     │
│         └∊──────────────┘                     │
└∊─────────────────────────────────────────────┘
```

syvyys (kerros, kerroksisuus), yhtenevyys (identtisyys) `≡K` `K1≡K2` {⎕CT}

Sääntiön sisäkkäisyyden asteen kertoo monadinen syvyysfunktio *(depth)*, jonka palauttama tulos on **skalaari**. Paljaalle skalaarille syvyys on `0`, paljaalle sääntiölle `1` ja sisäkkäiselle sääntiölle syvyys on = `1` + **sisäkkäisimmän** alkion syvyys.

```
          ≡¨'A' 'APL1' ('APL' 2) (⊂⊂⊂⊂'APLW')
0 1 2 5
```

Dyadinen yhtenevyysfunktio *(match)* ilmaisee kahden sääntiön keskinäisen **identtisyyden** eli sisällön ja rakenteen (muodon ja syvyyden) yhdenmukaisuuden.

```
          'A'≡,'A'
0
```

kätkentä (kapselointi), ositus (jaotus) `⊂[W]K` `⊂K` `V⊂[S]K` `V⊂K`

Monadinen kätkentäfunktio *(enclose)* paketoi (kapseloi) argumenttinsa sisäkkäiseksi sääntiöksi vektorin `W` osoittamissa suunnissa. Jos suuntamerkintä puuttuu, oletuksena ovat kaikki suunnat eli koko sääntiö kätketään **skalaariksi** (`⊂K ←→ ⊂[⍳⍴⍴K]K`).

Paljasta skalaaria **ei voi** kätkeä, eli paljaalle skalaarille `S`: `S ←→ ⊂S`.
Kätkentä kasvattaa argumenttinsa (muun kuin paljaan skalaarin) syvyyttä yhdellä.

```
      ⊂'ABC'
 ABC                                      ⍝ huomaa sisennys
      ≡⊂'ABC'
2
      A←3 4⍴'KÄSIUPOSNYSÄ' ⋄ ⊂[2]A
 KÄSI  UPOS  NYSÄ
      ⊂[1]A
 KUN  ÄPY  SOS  ISÄ
```

Jos koordinaattivektori `W` on tyhjä ja argumentti on sisäkkäinen sääntiö, sen alkioiden sisäkkäisyys kasvaa yhdellä **muun rakenteen** muuttumatta: `⊂[⍳0]N ←→ ⊂¨N`.

Kätkentä- ja paljastusfunktioiden välillä on yhteys (`W` ei saa olla tyhjä): `K ←→ ⊃[W]⊂[W]K`.
Vektori-ilmaisun ja kätkentäfunktion välillä on identiteetti: `A B C ←→ (⊂A),(⊂B),(⊂C)`.

Dyadisella ositusfunktiolla *(partition)*, jonka vasen ohjausargumentti `V` on skalaari tai kokonaislukuvektori (≥0), oikea argumentti `K` jaotellaan sisäkkäiseksi vektoriksi suunnassa `S` ohjausvektorin **jakokohtien** mukaan.
Jos ositussuuntamerkintä puuttuu, on oletuksena sääntiön viimeinen suunta.

Jakokohdassa ohjaimen alkion arvo **kasvaa** suhteessa edelliseen arvoon. Nollalla merkittyjä osia ei oteta tulokseen mukaan. Ohjausvektorin (ellei se ole skalaari) pituuden ja ositettavan sääntiön kokovektorin vastaavan arvon tulee olla yhtä suuret (`⍴V ←→ (⍴K)[S]`).

Tulokselle on voimassa: `(V≠0)/[S]K ←→ ⊃[S],/[S]V⊂[S]K`.

```
      DISPLAY 1 1 2 2 0 3 2 1 2 3⊂'ABCDEFGHIJ'
┌→─────────────────────────────────────┐
│ ┌→─┐ ┌→─┐ ┌→──┐ ┌→┐ ┌→┐ │
│ │AB│ │CD│ │FGH│ │I│ │J│ │
│ └──┘ └──┘ └───┘ └─┘ └─┘ │
└∊─────────────────────────────────────┘
      A←' VÄLILYÖNNEIN  EROTELTU    TEKSTI '
      DISPLAY (A≠' ')⊂A
┌→──────────────────────────────────────────┐
│ ┌→──────────┐ ┌→───────┐ ┌→─────┐ │
│ │VÄLILYÖNNEIN│ │EROTELTU│ │TEKSTI│ │
│ └────────────┘ └────────┘ └──────┘ │
└∊──────────────────────────────────────────┘
      1 0 1⊂[1]3 3⍴'ABCDEFGHI'
 A  B  C
 G  H  I
      1 0 1⊂[2]3 3⍴'ABCDEFGHI'
 A  C
 D  F
 G  I
```

paljastus (purku, kuorinta), poiminta ⊃[*W*]*K* ⊃*K* *V*⊃*K* {⎕*IO*}

Monadinen paljastusfunktio *(disclose, mix)* purkaa argumenttinsa alkiot sääntiöiksi **alkioiden** purkusuunnissa *W*. Kaikkien *K*:n alkioiden tulee olla skalaareja tai samanulotteisia sääntiöitä (ei välttämättä samanmuotoisia). Tulossääntiötä täydennetään tarvittaessa täytealkioilla.

Suuntamerkinnän puuttuessa oletuksena ovat alkioiden **viimeiset** suunnat: ⊃*K* ←→ ⊃[(⍴⍴*K*)+⍳⍴⍴↑*K*]*K*. Paljastuksen ja kätkennän välillä on identiteetti: *K* ←→ ⊃[*W*]⊂[*W*]*K*.

```
        ⊃'SISÄKKÄINEN' 'VEKTORI' '' 'MATRIISIKSI'
SISÄKKÄINEN
VEKTORI

MATRIISIKSI
        ⊃(1 2)'ABCD' 'EFGHI'
1 2 0 0 0
A B C D
E F G H I
```

Tuloksen rakenteessa yhdistyvät alkuperäisen sisäkkäisen sääntiön ja sen alkioiden rakenteet. Purkusuuntavektorin *W* alkiot määrittävät alkion suuntien sijainnit **tuloksen** kokovektorissa.

Esimerkiksi jos ⍴*K* ←→ 3 ja ⍴¨*K* ←→ (2 4)(2 4)(2 4), on ⍴⊃[1 2]*K* ←→ 2 4 3, ⍴⊃[1 3]*K* ←→ 2 3 4 ja ⍴⊃[2 3]*K* ←→ 3 2 4.

Jos kaikki *K*:n alkiot ovat skalaareja, täytyy *W*:n olla tyhjä.

```
        ⊃[⍳0]⊂¨'AB' 'CDE' 'FGHI'
 AB CDE FGHI
```

Dyadisella poimintafunktiolla *(pick)* voi valita ja paljastaa **minkä tahansa**, sisäkkäisenkin, sääntiön alkion. Valintavektorille *V* ovat voimassa ehdot: 2≥≡*V* ja (⍴*V*)≤≡*K*. *V*:n alkiot määrittävät *K*:n vastaavan kerroksen (syvyyden) indeksoinnin.

```
        A←'A' 'BC' ('DEF' 'GHIJ') ⋄ 3⊃A
 DEF  GHIJ
        3 2 4⊃A                     ⍝ ←→ 4⊃2⊃3⊃A
J
```

Moniulotteisen sääntiön uloimmasta kerroksesta (kuorikerroksesta) valittaessa *V*:n tulee olla skalaari tai yksialkioinen vektori. Tyhjällä vektorilla valittaessa: *K* ←→ (⍳0)⊃*K* (myös skalaarille).

```
        A←2 3⍴'AB' 'CDE' 'FGHI' ('JK' 'LMN') 'O' 'PQ'
        (⊂2 1)⊃A
 JK  LMN
        (2 1)2⊃A
LMN
        (I J)←((1 2)3) ((2 1)2 3)
        I J⊃¨⊂A
EN
        (⍳0)⊃2*0.5
1.414213562
```

ensimmäinen (eka), otto `↑K     V↑[W]K     V↑K`

Monadinen ekafunktio *(first)* valitsee argumenttinsa ensimmäisen alkion ja **paljastaa** sen.
Jos *K* on tyhjä, tuloksena on *K*:n **prototyyppi**.

Ei-tyhjälle sääntiölle *K*: `↑K ←→ (⊂(ρρK)ρ1)⊃K`.
Paljaalle vektorille *V*: `↑V ←→ ''ρV ←→ 1⊃V ←→ 1⌷V`.

```
          ↑'DO' 'RE' 'MI' 'FA'
DO
          ↑↑'DO' 'RE' 'MI' 'FA'
D
```

Dyadisen ottofunktion *(take)* vasen kokonaislukuvektoriargumentti *V* ilmaisee suunnissa *W* otettavien alkioiden lukumäärän. Vektorissa *V* on oltava suuntien verran alkioita. Suuntamerkinnän puuttuessa ovat oletuksena kaikki *K*:n suunnat (`V↑K ←→ V↑[ιρρK]K`). Suuntavektorin *W* alkioiden järjestys on vapaa, mutta kukin suunta voidaan antaa vain kerran.

Positiiviselle *V*:n alkiolle valitaan vastaavan suunnan alkiot alkupäästä ja negatiiviselle lopusta. Jos jossain suunnassa valitaan enemmän alkioita kuin niitä *K*:ssa on, ajatellaan *K*:n sisältävän ylimääräisiä tasoja, jotka kaikki sisältävät *K*:n vastaavan suunnan prototyyppejä.

Otolle on voimassa identiteetti: `V↑[W]K ←→ ⊃[W](⊂V)↑¨⊂[W]K`.
Otto **ei** vaikuta valittujen alkioiden sisäiseen rakenteeseen.

```
          A←⊃'PUUPÄÄ' 'ETUOVI' 'JÄÄNYT'
          2↑[1]A                      ⍝ ←→ 2 6↑A
PUUPÄÄ
ETUOVI
          ¯3↑[2]A                     ⍝ ←→ 3 ¯3↑A
PÄÄ
OVI
NYT
          3 ¯2↑[2 1]A                 ⍝ ←→ ¯2 3↑A
ETU
JÄÄ
          3↑[1]⊃(1 'A' 3)(4 5 6)
1 A 3
4 5 6
0   0
```

pudotus `V↓[W]K     V↓K`

Pudotusfunktio *(drop)* toimii samalla tavoin kuin otto, mutta vasen kokonaislukuargumentti *V* ilmaisee nyt suunnittain **pois** jätettävien alkioiden lukumäärät.

```
          2↓[1]A                      ⍝ ←→ 2 0↓A
JÄÄNYT
          3 ¯1↓[2 1]A
PÄÄ
OVI
```

jonoutus ,[W]K ,K

Monadisella jonoutusfunktiolla *(ravel)* argumentin K alkiot jonoutetaan suunnissa W. Suuntavektori W voi olla joko kokonaislukuvektori, -skalaari, desimaaliluku tai tyhjä vektori. Suuntamerkinnän puuttuessa on oletuksena sääntiön kokovektori: ,K ←→ ,[⍳⍴⍴K]K.

Jos W on desimaaliluku (1⌈W⌊1+⍴⍴K), ajatellaan argumenttiin lisätyksi uusi suunta ⌈W, jota vastaavaksi kokovektorin alkioksi tulee 1.

```
          A←2 3⍴'MIESIE'                  ⍝ ←→ ⊃'MIE' 'SIE'
          ,[0.1]A
MIE
SIE
          ⍴,[0.1]A
1 2 3
          ⍴,[1.1]A
2 1 3
          ⍴,[2.1]A
2 3 1
```

Jos W on kokonaislukuvektori, sen tulee olla **nousevassa suuruusjärjestyksessä** oleva osakoordinaatti-vektori. Tuloksen kokovektoriin **tuloutuvat** tällöin W:n sisältämät K:n suunnat.
Kokonaisluskuskalaarille W: K ←→ ,[W]K.

```
          A←3 2 4⍴⍳24 ⋄ ,[2 3]A          ⍝ ⍴,[2 3]A ←→ 3,(2×4) ←→ 3 8
 1  2  3  4  5  6  7  8
 9 10 11 12 13 14 15 16
17 18 19 20 21 22 23 24
```

Jos W on tyhjä vektori, muodostetaan kokovektorin **loppuun** uusi ulottuvuus, jonka pituus = 1.

```
          ,[⍳0]11 12                      ⍝ ←→ ,['']11 12
11                                        ⍝ ⍴,[⍳0]11 12 ←→ 2 1
12
```

paitsi (ilman) V~K

{⎕CT}

Dyadinen paitsi-funktio *(without, excluding)* poistaa vasemmasta vektoriargumentistaan V oikeassa argumentissa K olevat alkiot. Poistetaan vain ne alkiot, jotka ovat **sekä** rakenteeltaan **että** sisällöltään yhtenevät.

```
          1 1 2 2 3 3 4 4 5 5~9 5 7 3 1
2 2 4 4
          'HIP' 'HIP' 'HURRAA'~'HIPHIP' 'HURRAA'
 HIP  HIP
          4 7 5(⍳0)6 7 5~9 5 7 3
4   6                                     ⍝ ⍳0 jäi!
          4 7 5(⍳0)6 7 5~9 5 7 3,⊂⍳0
4 6
          'ABC'~2 2⍴'BDEF'                ⍝ poistettavan muodolla ei väliä
AC
```

nousuindeksi ⍋K   K1⍋K2 {⎕IO}

Monadisen nousuindeksin *(grade up)* ⍋K tuloksena on vektori, jossa **numeerisen** sääntiön K **ensimmäisen** suunnan indeksivektori on K:n alkioiden mukaisessa nousevassa suuruusjärjestyksessä.

Esimerkiksi matriisiargumentilla tuloksena on vektori, joka osoittaa mihin järjestykseen matriisin rivit järjestettäisiin nousevasti.

```
      M←3 2⍴2 1,1 1,1 2 ◇ ⍋M      ⍝ M ←→ ⊃(2 1)(1 1)(1 2)
2 3 1
      DISPLAY¨ M (M[⍋M;])
┌→──┐ ┌→──┐
↓2 1│ ↓1 1│
│1 1│ │1 2│
│1 2│ │2 1│
└~──┘ └~──┘
```

Dyadista nousuindeksiä *(grade up with collating sequence)* K1⍋K2 käytetään **merkkisääntiöiden** lajitteluun. Vasen ohjainargumentti K1 sisältää sen aakkostusjärjestyksen, jonka mukaan K2:n ensimmäisen suunnan indeksivektorin järjestys määrätään.

Esimerkiksi jos K1 on vektori ja K2 on matriisi, lasketaan K2:n riveille painoarvot niiltä löytyvien K1:n merkkien mukaan ja näillä painoarvoilla tuotetaan edelleen tulosindeksivektori. Jos K1 on matriisi tai moniulotteinen sääntiö, samoilla riveillä ovat ne merkit joilla on sama painoarvo; moniulotteisella ohjaimella voi tarvittaessa hienosäätää useiden samanpainotteisten merkkien keskinäistä järjestystä.

```
      A1←'ABCDEFGHIJ'
      A2←⊃'ABCDEFGHIJ' 'abcdefghij'
      M←⊃'ABC' 'abc' 'AbC' 'Abc' 'ABc' ' BC' ' AB'
      DISPLAY¨ M (M[A1⍋M;]) (M[A2⍋M;]) (M[⎕AV⍋M;])
┌→──┐ ┌→──┐ ┌→──┐ ┌→──┐
↓ABC│ ↓ABC│ ↓ABC│ ↓ AB│
│abc│ │ABc│ │ABc│ │ BC│
│AbC│ │AbC│ │AbC│ │abc│
│Abc│ │Abc│ │Abc│ │Abc│
│ABc│ │ AB│ │abc│ │AbC│
│ BC│ │ BC│ │ AB│ │ABc│
│ AB│ │abc│ │ BC│ │ABC│
└───┘ └───┘ └───┘ └───┘
```

laskuindeksi ⍒K   K1⍒K2 {⎕IO}

Monadinen ja dyadinen laskuindeksi *(grade down; grade down with collating sequence)* toimivat samalla tavoin kuin nousuindeksifunktiot, paitsi että indeksin järjestys on nyt **laskeva**.

```
      DISPLAY¨ M (M[A1⍒M;]) (M[A2⍒M;]) (M[⎕AV⍒M;])
┌→──┐ ┌→──┐ ┌→──┐ ┌→──┐
↓ABC│ ↓abc│ ↓ BC│ ↓ABC│
│abc│ │ BC│ │ AB│ │ABc│
│AbC│ │ AB│ │abc│ │AbC│
│Abc│ │abc│ │Abc│ │Abc│
│ABc│ │AbC│ │AbC│ │abc│
│ BC│ │ABc│ │ABc│ │ BC│
│ AB│ │ABC│ │ABC│ │ AB│
└───┘ └───┘ └───┘ └───┘
```

etsintä K1⍷K2

{⎕CT}

Etsintäfunktiolla *(find)* paikallistetaan `K2`:sta `K1`:n kanssa yhteneviä (samanmuotoisia ja -sisältöisiä) alueita. Loogisessa tulossääntiössä on ykkönen jokaisen löytyneen alueen **ensimmäisen** (kulma-)alkion paikalla. Tulossääntiö on samanmuotoinen kuin `K2`.

```
         'SIKA'⍷'UUSIKAARLEPYY'
0 0 1 0 0 0 0 0 0 0 0 0 0
         A←'*   TIIVISTETTÄVÄ   TEKSTI   *'
         (~'  '⍷A)/A        ⍝ peräkkäisten välilyöntien tiivistys
* TIIVISTETTÄVÄ TEKSTI *
         A←'TOM' 'TOM' 'TO' 'TOM' 'TOM' 'TOM'
         'TOM' 'TOM'⍷A
1 0 0 1 1 0
         (⊂'TOM')⍷A
1 1 0 1 1 1
```

Jos `K1`:llä on vähemmän suuntia kuin `K2`:lla, etsitään `K2`:n **viimeisten** suuntien mukaisesti eli `K1`:n kokovektorin **alkuun** lisätään tällöin tarpeellinen määrä ykkösiä.

```
         A←4 5⍴'ABCABA'
         DISPLAY¨ A ('C'⍷A) ('AB'⍷A) ((2 2⍴'BCAB')⍷A)
┌→────┐  ┌→────────┐  ┌→────────┐  ┌→────────┐
↓ABCAB│  ↓0 0 1 0 0│  ↓1 0 0 1 0│  ↓0 1 0 0 0│
│AABCA│  │0 0 0 1 0│  │0 1 0 0 0│  │0 0 1 0 0│
│BAABC│  │0 0 0 0 1│  │0 0 1 0 0│  │0 0 0 1 0│
│ABAAB│  │0 0 0 0 0│  │1 0 0 1 0│  │0 0 0 0 0│
└─────┘  └~────────┘  └~────────┘  └~────────┘
```

mallimuotoilu (esimerkkimuotoilu) V⍕K

{⎕FC}

Dyadisen mallimuotoilufunktion *(format by example)* vasempana argumenttina on merkkivektori, joka sisältää muotoilua ohjaavat erikoismerkit `0..9` ja `.` sekä `,`.

Muut merkkivektorin sisältämät merkit säilytetään tuloksessa sellaisinaan.

Muotoiluohjaimet toimivat lakonisesti kerrottuna seuraavasti:

.	desimaalierottimen sijainti (numerojonon osana)
,	tuhaterottimen sijainti (numerojonon osana)
0	täytä nollilla tähän asti
1	etumerkkimerkintä negatiiviselle luvulle
2	etumerkkimerkintä positiiviselle luvulle
3	käytä etumerkkimerkintää (yhdessä ohjainten `1` tai `2` kanssa)
4	vaihda ohjainten `1`, `2` tai `3` toiminta päinvastaiseksi
5	standardi lukumuotoilu
6	kenttäerottimen käsittely
7	eksponenttimuodon ilmaisin
8	välilyöntipositioiden täytönilmaisin
9	täytä nollilla tai välilyönneillä tähän asti.

```
         ('50. 50. 5555'⍕⎕TS[3 2 1])~' '      ⍝ päiväys
8.8.1987
         'HINTA: 5550.50⎕'⍕123.45
HINTA:  123.45⎕
```

Tärkeimpiä APL2-järjestelmämuuttujia ja -funktioita

⎕*AF* *(Atomic Function)* Koodivektoriavain: muunnetaan argumentin kokonaislukuindeksit vastaaviksi ⎕*AV*:n (nolla-alkuisiksi) merkeiksi; merkkiargumentit muunnetaan vastaaviksi indekseiksi.

⎕*AT* *(Attributes)* Lisätiedot: palautetaan oikean argumentin nimilista objekteista vasemman ohjainargumentin mukaisia tietoja. Vasen ohjainargumentti voi olla joku kokonaisluku `1-4`.

⎕*EA* *(Execute Alternate)* Ehdollinen suoritus (dyadinen): suorittaa funktion oikean tekstiargumentin. Virhetilanteessa suoritetaan **vasen** tekstiargumentti.

⎕*EC* *(Execute Controlled)* Hallittu suoritus (monadinen): suorittaa funktion tekstiargumentin.
Tuloksena sisäkkäinen vektori: (lauseketyyppi) (paluukoodi) (lausekkeen tulos).

⎕*EM* *(Error Message)* Virheviesti: kolmirivinen tekstimatriisi.

⎕*FC* *(Format Control)* Muotoilumerkistö: tekstivektori, jossa ovat muotoilussa tulostuvat desimaalierotin, tuhaterotin, täyte-, ylivuoto-, välilyönti- ja negatiivisuusmerkki. Oletusarvona `'.,*0_¯'`.

⎕*NC* *(Name Classification)* Nimiluokka: kertoo argumenttinsa (merkkimatriisi tai -vektori) objektiluokan (`¯1/0/1/2/3/4` = väärä syntaksi/ei käytössä/riviosoite/muuttuja/funktio/operaattori).

⎕*NL* *(Name List)* Nimiluettelo: oikea argumenttiskalaari/vektori ilmaisee sen objektiluokan, josta etsitään (`1`= riviosoitteet, `2` = muuttujat, `3` = funktiot, 4 = operaattorit).

⎕*TF* *(Transfer Form)* Siirtomuoto: argumenttiobjekti siirtotiedoston mukaiseksi tekstivektoriksi tai muunnos siirtomuodosta APL-objektiksi.

⎕*TC* *(Terminal Control)* Erikoismerkkivektori: askelpalautus-, rivinvaihto- ja rivinsiirtomerkki.

Tärkeimpiä APL2-järjestelmäkomentoja

Osalle järjestelmäkomennoista voi APL2:ssa antaa sen nimivälin, jota komento koskee.
Esimerkiksi `)VARS A C` tuo esille ne muuttujat, jotka alkavat kirjaimilla *A*, *B* tai *C*.

`)HOST {cmd}`
: Suorita käyttöjärjestelmäkäsky `cmd`, tai siirry väliaikaisesti käyttöjärjestelmätasolle.

`)IN atf {obj1 obj2 ..}`
: Lue siirtotiedosto `atf` joko kokonaan tai objektit `obj1...`

`)NMS {C {E}}`
: Listaa työtilan muuttujat, funktiot ja operaattorit (jotka alkavat kirjaimilla `C..E`).

`)OUT atf {obj1 obj2 ..}`
: Vie siirtotiedostoksi `atf` joko koko työtila tai objektit `obj1,...`

`)OPS {C {E}}`
: Listaa työtilan operaattorit (jotka alkavat kirjaimilla `C..E`).

`)PIN atf {obj1 obj2 ..}`
: Kuten `)IN`, mutta haetaan vain ne objektit, joiden nimiä ei ole aktiivisessa työtilassa käytössä (*Protected* `IN`).

APL2-idiomeja

```
,[0.1]V                                 ⍝ vektorista yksirivinen matriisi
,['']V                                  ⍝ vektorista pystymatriisi
,[⍳⍴⍴K],[0.1]K                          ⍝ sääntiöstä matriisi (vektorista vaakamatriisi)
,[¯1↓⍳⍴⍴K]K                             ⍝ sääntiöstä matriisi (vektorista pystymatriisi)
,/M                                     ⍝ matriisin riveistä sisäkkäinen vektori
,⌿M                                     ⍝ matriisin sarakkeista sisäkkäinen vektori
V/W                                     ⍝ vektori (V[1]⍴W[1]),(V[2]⍴W[2]),..
2≠/V                                    ⍝ sama-alkioisten ryhmien alkumerkit
¯2-/V                                   ⍝ vierekkäisten alkioiden erotukset
(1+V='''')/V                            ⍝ heittomerkkien kahdennus merkkivektorista
(~'  '⍷V)/V                             ⍝ peräkkäisten välilyöntien tiivistys yhdeksi
V~' '                                   ⍝ välilyöntien poisto vektorista
V~V~W                                   ⍝ vektorijoukkojen leikkaus
V,W~V                                   ⍝ vektorijoukkojen unioni
↑0⍴⊂K                                   ⍝ tyyppisääntiö
↑0⍴⊂↑K                                  ⍝ prototyyppi
V⊃¨⊂K                                   ⍝ hajaindeksipoiminta ("chipmunk")
⊂[2]M                                   ⍝ matriisista sisäkkäinen rivivektori
(⊂[2]M)~¨' '                            ⍝ matriisin riveistä välilyönnitön sisäkkäinen vektori
(V≠' ')⊂V                               ⍝ vektorin ositus välilyöntien mukaan
⊂[1↓⍳⍴⍴K]K                              ⍝ ei-skalaarin vektorointi
K≡¨⊂V                                   ⍝ mitkä sisäkkäisen sääntiön solut ovat = vektori V
⊃⍎¨⊂[2]M                                ⍝ merkkimatriisista numeerinen
⊃⍎¨⊂[⍴⍴K]',',K                          ⍝ merkkisääntiö riveittäin numeeriseksi
⊃(⊂¨⍋¨A)⌷¨A←⊂[⍴⍴A]A←1/A                 ⍝ sääntiön rivit nousevaan järjestykseen
⊃V                                      ⍝ sisäkkäinen vektori (rivi)matriisiksi
⍉⊃V                                     ⍝ sisäkkäinen vektori (sarake)matriisiksi
⊃,/K1 K2                                ⍝ sääntiöiden liitos riveittäin
,¨/K1 K2                                ⍝ sääntiöiden alkioittainen liitos
M⌊.⍳' '                                 ⍝ matriisin riveiltä löytyvien välien ensimmäiset indeksit
I⌷¨⊂K                                   ⍝ sääntiön hajaindeksointi sisäkkäisellä indeksivektorilla I
(⊂⊂W)⌷[1]¨V                             ⍝ matriisivektorin alkioiden rivilajittelu indeksein W
(,M)←(,M)[⍋,(2×⍳↑⍴M)+[1]M=' ']          ⍝ siirrä välilyönnit matriisin rivien loppuun
((,K='¯')/,K)←'-'                       ⍝ vaihda sääntiössä negatiivit miinusmerkeiksi
1⊃4 ⎕AT K                               ⍝ muuttujan koko tavuina
2 ⎕AT F                                 ⍝ funktion päivitysaikaleima
```

Kirjallisuutta

Osa seuraavista APL-julkaisuista on ollut suoranaisina lähteinä, osa on muuten vaan suositeltavaa APL-luettavaa asiasta syvää tietoa halajaville.

Seppo **Linnainmaan**, Heikki **Apiolan**, Kyösti **Huhtalan** ja Tapio **Nummen** luentomonisteet.

APL-konferenssien esitelmäjulkaisut.

Gary A. **Bergquist**:
APL Advanced Techniques and Utilities, 1987

Per **Gjerløv**, Henrik E. **Nyegaard** (suomeksi kääntänyt Gustav **Tollet**):
APL-kielen opas, 1975 (IBM G075-0009-1F)

Leonard **Gilman**, Allen J. **Rose**:
APL – an Interactive Approach, 1983 (ISBN 0-471-09304-1)

FinnAPL:
Idiomikirjasto, 1984 (ISBN 951-95886-0-4)

James A. **Brown**:
The Principles of APL2, 1984 (IBM TR 03.247)

IBM:
An Introduction to APL2, 1988 (IBM SH20-9229-1)
APL2 Programming: Language Reference, 1988 (IBM SH20-9227-3)

James A. **Brown**, Sandra **Pakin**, Raymond P. **Polivka**:
APL2 at a Glance, 1988 (ISBN 0-13-038670-7)

Juha **Haataja**:
APL2-ohjelmointikielen käyttöopas, 1989

Norman D. **Thomson**, Raymond P. **Polivka**:
APL2 in Depth, 1995 (ISBN 0-387-94213-0)

Lisäksi lehtiä:

Suomen APL-yhdistys:	*APL-Uutiset*
British APL Association:	*Vector*
ACM/SigAPL:	*Quote Quad*

WWW-osoitteita

Suomen APL-yhdistys	http://www.pyr.fi/APL/
British APL Association	http://www.vector.org.uk
ACM: SigAPL	http://www.acm.org/sigAPL/
APL Journal	http://www.rhombos.de/apljourn.htm
Association Francophone pour la promotion d'APL	http://www.afapl.asso.fr/
Waterloo APL Archive	ftp://watserv1.uwaterloo.ca/languages/apl/index.html
Open Directory: APL	http://dmoz.org/Computers/Programming/Languages/APL/
APL FAQ	http://www.izap.com/~Esirlin/apl/apl.faq.html
IBM (APL2)	http://www-3.ibm.com/software/ad/apl/support.html
Dyadic (DyalogAPL)	http://www.dyadic.com
APL2000 (APL+)	http://www.apl2000.com/
MicroAPL (APLX)	http://www.microapl.co.uk/
APL2C	http://www.apl2c.de/home/
Causeway	http://www.causeway.co.uk/
Soliton (SharpAPL)	http://www.soliton.com/
A+	http://www.aplusdev.org/
J	http://www.jsoftware.com/
K	http://www.kx.com/

APL2-perusnäppäimistö