

Algoritmisen ajattelun peruskäsitteitä: sekvensointi, valinta ja toisto

Sisällysluettelo

[Johdanto 3](#_heading=h.30j0zll)

[Mikä on algoritmi? 3](#_heading=h.1fob9te)

[Sekvensointi 4](#_heading=h.3znysh7)

[Valinta 6](#_heading=h.tyjcwt)

[Toisto 7](#_heading=h.3dy6vkm)

[Pseudokoodi 9](#_heading=h.1t3h5sf)

[Esimerkkitehtävät 12](#_heading=h.4d34og8)

[Viitteet](#_heading=h.9bmp1pwq9i7i) 13

# Johdanto

Algoritmisella ajattelulla tarkoitetaan ongelmanratkaisukykyä, jolla pystytään määrittämään jokin selkeä ja tarkasti rajattu vaiheiden ketju. Kun ketjua noudatetaan tarkasti, sen avulla voidaan löytää ratkaisu ongelmaan. Koulussa, ja eritoten peruskoulun ja lukion aikana, opimme ratkaisemaan ongelmia muistinvaraisen oppimisen avulla sekä jäljittelemällä opettajan antamia ratkaisuja. Vaikka esimerkiksi monet oppilaat osaavat laskea kahden luvun suurimman yhteisen nimittäjän, vain harvat pystyvät sanoittamaan ääneen niiden toimintojen luettelon, jota tarvitaan oikean tuloksen saamiseksi.

Ratkaisun löytämiseksi tarvittavien vaiheiden määrittely on kuitenkin erittäin tärkeää, koska sen lisäksi, että se mahdollistaa tehtävien automatisoinnin (esim. luomalla koneen, joka pystyy suorittamaan halutut vaiheet), vaiheiden määrittelyn avulla yksilö tulee tietoiseksi ongelmasta, sen ratkaisusta ja siten kyvystä mukauttaa löydetty ratkaisu samanlaisiin tai vastaaviin ongelmiin: jotta voidaan määritellä selkeät ja tarkkaan määritellyt ohjeet, on ensin ymmärrettävä itse ongelma, olosuhteet, joista lähdetään liikkeelle ja joihin halutaan päästä sekä välineet ja toiminnot, joiden avulla järjestelmää voidaan muuttaa ohjelmoidusti ja ennustettavasti.

Vaikka käsitteet "algoritmi" ja "algoritminen ajattelu" liitetään usein tietotekniikkaan, ne ovat välttämättömiä työkaluja jokapäiväisessä elämässämme. Ajatellaanpa vaikka ruoanlaittoa ja reseptejä, joita käytetään minkä tahansa ruoan valmistamiseen, kirurgista toimenpidettä tai rakennuksen rakentamista. Algoritmien määrittely on yksi niistä prosesseista, joita käytetään eniten tiedon ja määriteltyjen prosessien välittämiseen.

Tässä tekstissä tarkastellaan algoritmin käsitettä yksityiskohtaisesti. Sen lisäksi tarkastellaan algoritmeihin liittyviä tärkeimpiä käsitteitä, kuten sekvensointia, valintaa ja toistoa.

# Mikä on algoritmi?

Algoritmi, jota kutsutaan usein myös menettely- tai käsittelyjärjestykseksi, on joukko selkeitä ja tarkkaan määriteltyjä vaiheita, joiden avulla voidaan suorittaa jokin tehtävä. Jos haluamme ymmärtää, miten algoritmi toimii, voimme käyttää esimerkkinä reseptiä, jonka avulla voi keittää kahvia mokkapannulla. Reseptin sisältö voidaan esittää yksinkertaisesti seuraavalla tavalla[[1]](#footnote-0):

* Pura mokkapannu kolmeen pääosaan: alaosaan, suodattimeen ja yläosaan.
* Kaada vettä mokkapannun alaosaan.
* Aseta metallinen suodatin suodatinkoloon mokkapannun alaosan päälle.
* Täytä suodatin kahvijauheella.
* Kierrä mokkapannun yläosa sen alaosan päälle.
* Aseta mokkapannu liedelle.
* Kytke liesi päälle.
* Odota, että kahvi valuu ulos ja mokkapannun yläosa täyttyy.
* Sammuta liesi.

Kuka tahansa voi helposti seurata näitä ohjeita; jopa melko kehittynyt robotti onnistuu siinä. Nämä ohjeet johtavatkin aina samaan lopputulokseen: hyvään kahviin.

Eli ovatko kaikki ohjeluettelot siis algoritmeja? Itse asiassa virallisesti algoritmille on aina ominaista neljä perusominaisuutta: yksiselitteisyys, ilmaisun äärellisyys, laskennan äärellisyys ja deterministisyys.

* 1. **Yksiselitteisyys**

Yksiselitteisyydellä tarkoitetaan sitä, että algoritmin toimeenpanijan on kyettävä ymmärtämään algoritmin kuvaus ja siten tunnistamaan kieli, jolla menettelyjärjestyksen muodostavat vaiheet on ilmaistu. Tällä valitulla kielellä tarkasti muotoiltuja lauseita kutsutaan muodollisesti ohjeiksi.

Toisin sanoen tämä ominaisuus tarkoittaa sitä, että jos algoritmi määritellään käyttämällä tiettyä kieltä, esimerkiksi englantia, mikä tahansa algoritmin toteuttaja, joka kykenee ymmärtämään kyseistä kieltä, tulkitsee ohjeet samalla tavalla.

* 1. **Ilmaisun äärellisyys**

Algoritmin kuvaama käsittelyjärjestys on esitettävä äärellisellä määrällä ohjeita. Ohjeiden lukumäärällä tai käsittelyjärjestyksen pituudella ei ole merkitystä (on myös olemassa käsittelyjärjestyksiä, joissa on hyvin suuri määrä ohjeita), kunhan ohjeiden määrä ei ole ääretön.

* 1. **Laskennan äärellisyys**

Samoin kuin lausekkeen äärellisyys, algoritmille on myös aina ominaista äärellinen määrä suoritusvaiheita. Toisin sanoen algoritmille on aina määriteltävä ehto, jonka täyttyessä suoritus päättyy. Tärkeää ei ole vaiheiden lukumäärä vaan itse suorituksen äärellisyys.

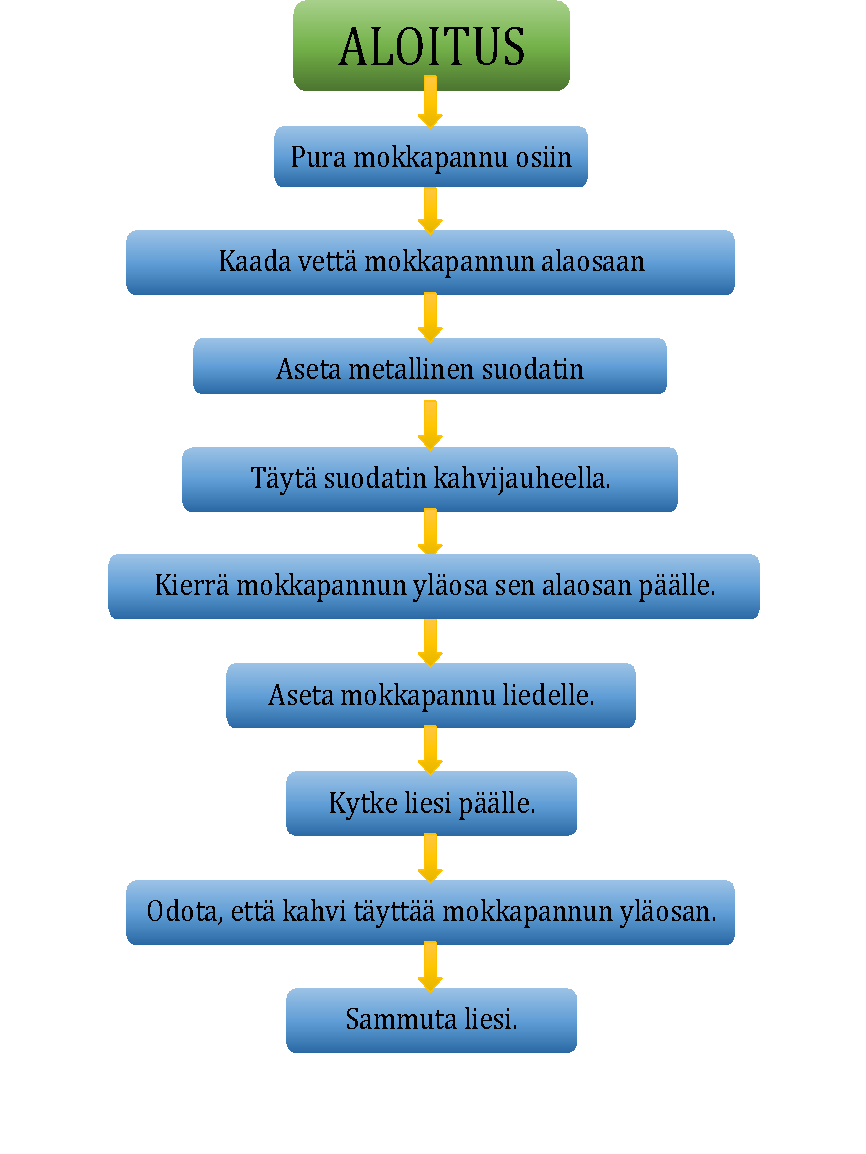
* 1. **Deterministisyys**

Algoritmin on oltava deterministinen, mikä tarkoittaa, että kun samaa algoritmia sovelletaan samaan syöttötietojoukkoon, saadaan aina samat tulokset.

Näiden neljän ominaisuuden valossa ymmärrämme, että jotta kahvireseptia voidaan pitää algoritmina on tehtävä tiettyjä oletuksia: puhtaan, kootun ja käyttövalmiin mokkapannun on aina oltava saatavilla, lieden on oltava aina saatavilla, vettä ja kahvijauhetta on oltava aina saatavilla. Entä jos haluaisimmekin tehdä algoritmistamme monipuolisemman, ottaa huomioon nämä alkuehdot ja yleistää niitä siten, että saisimme aina toteutettavissa olevan menettelytavan? Seuraavissa luvuissa tarkastelemme lähemmin joitakin työkaluja, joiden avulla voimme tarkentaa algoritmiamme ja samalla ymmärtää paremmin tätä maailmaa. Tarkastelemme erityisesti kolmea perusrakennetta: sekvensointia, valintaa ja toistoa. Nämä ovat itse asiassa algoritmin tärkeimmät perusrakenteet kiitos kahden tutkijan, Corrado Böhmin ja Giuseppe Jacopinin [1], 1960-luvulla määrittelemän teoreeman. Teoreeman mukaan mikä tahansa algoritmi voidaan toteuttaa käyttämällä vain sekvensointia, valintaa ja toistoa!

# Sekvensointi

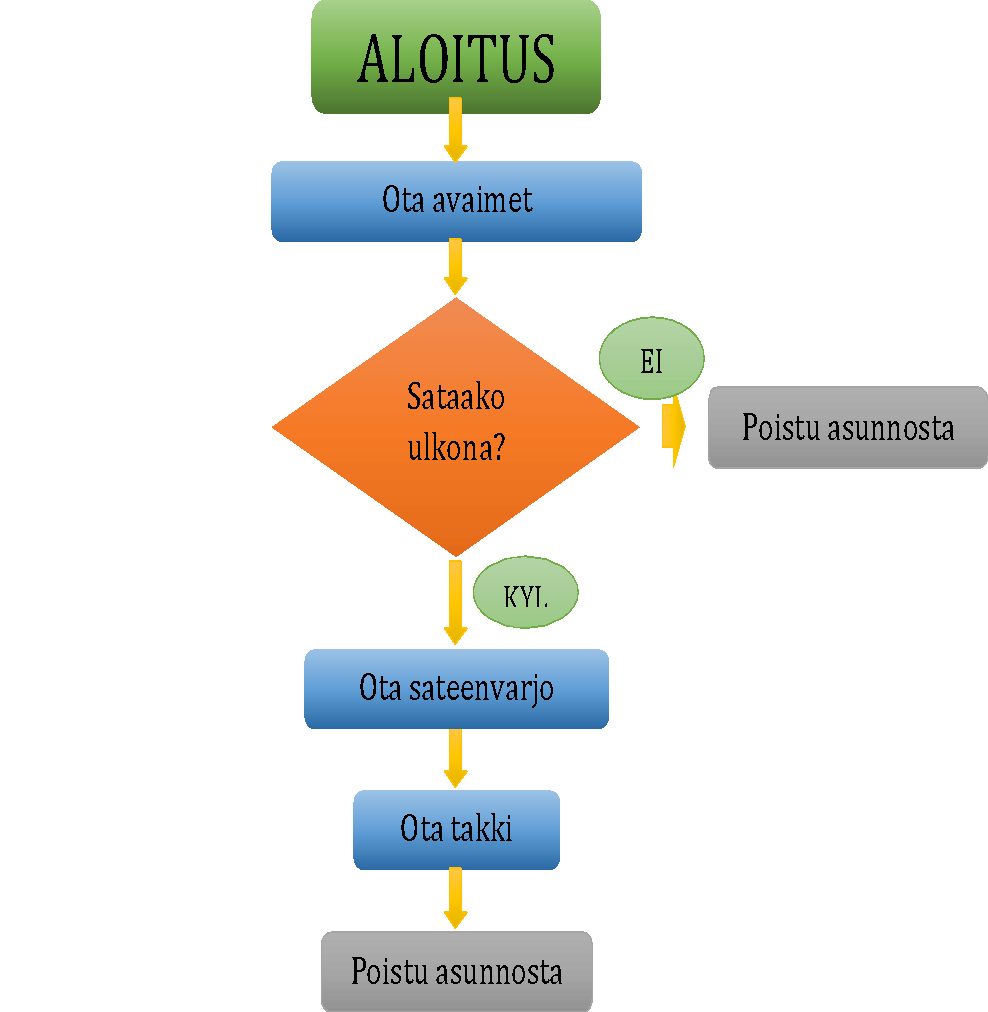
Sekvensoinnilla tarkoitetaan algoritmiin kuuluvien ohjeiden järjestyksen määrittämistä. Tarkastellaan esimerkiksi ohjeita, jotka muodostavat kahvireseptin:



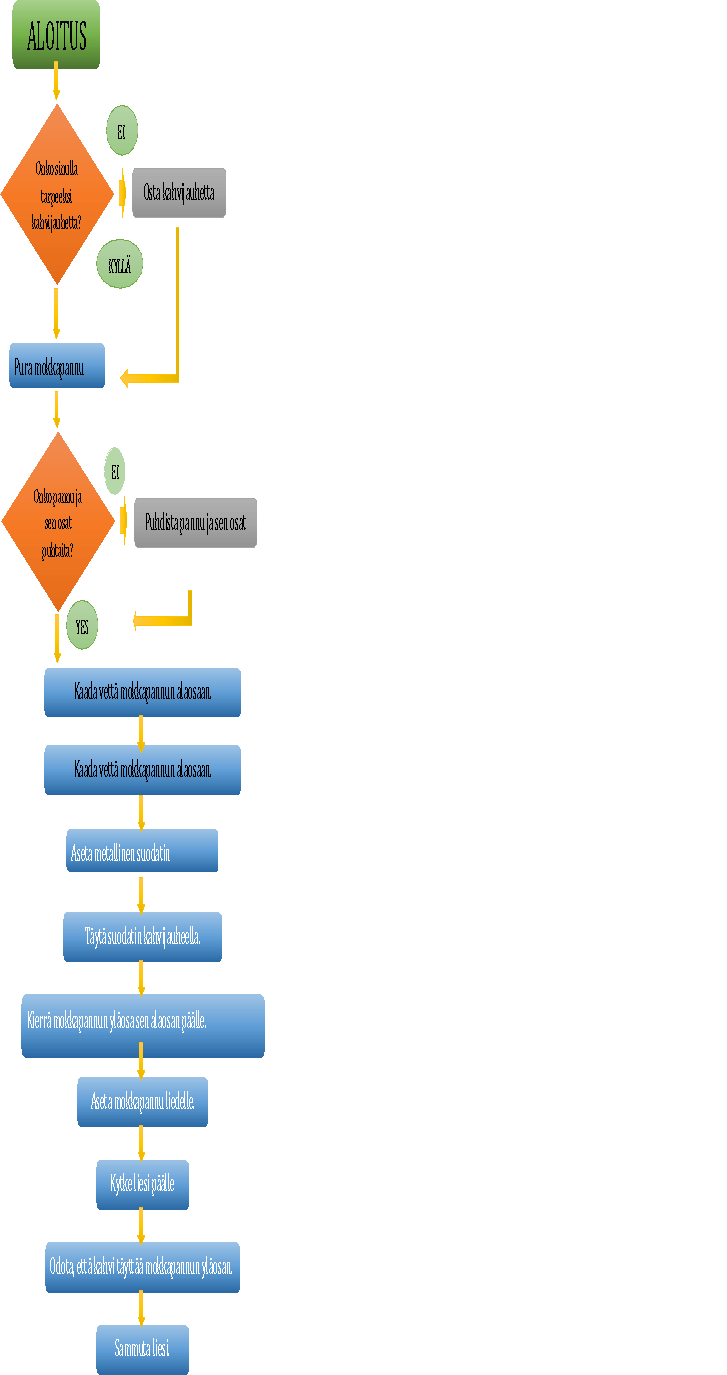
Jotta voidaan olla varmoja, että saavutetaan aina sama lopputulos (deterministisyys), jokainen vaihe on suoritettava määrätyssä järjestyksessä. Tämän esimerkin jälkeen on helppo ymmärtää, että vaiheiden järjestyksen muuttaminen johtaisi erilaiseen (ja väärään!) lopputulokseen: mitä esimerkiksi tapahtuisi, jos yrittäisit täyttää mokkapannun alaosan vedellä ennen kuin purat itse mokkakeittimen?

# Valinta

Valinta on rakenne, jonka avulla voimme määritellä erilaisia "suorituspolkuja" ja valita niistä yhden. Valinta riippuu tietyn ehdon vahvistamisesta (esim. onko ehto tosi vai epätosi?). Jos esimerkiksi luodaan algoritmi, jonka tarkoituksena on kerätä tärkeät esineet ennen kotoa lähtöä, voidaan sille määritellä kaksi polkua riippuen siitä, sataako ulkona vai ei:



Valinta on erittäin tehokas rakenne, jonka avulla algoritmista voidaan tehdä joustava ja mukautuva. Kun käytämme esimerkkinä kahvin keittämiseen tarkoitettua algoritmia, voimme ryhtyä testaamaan ehtoja varmistaaksemme, että sitä voidaan soveltaa yhä useammin arkitodellisuudessa: mitä tapahtuu, jos kahvipannu on likainen? Mitä tapahtuu, jos kahvijauhe on loppunut?



# Toisto

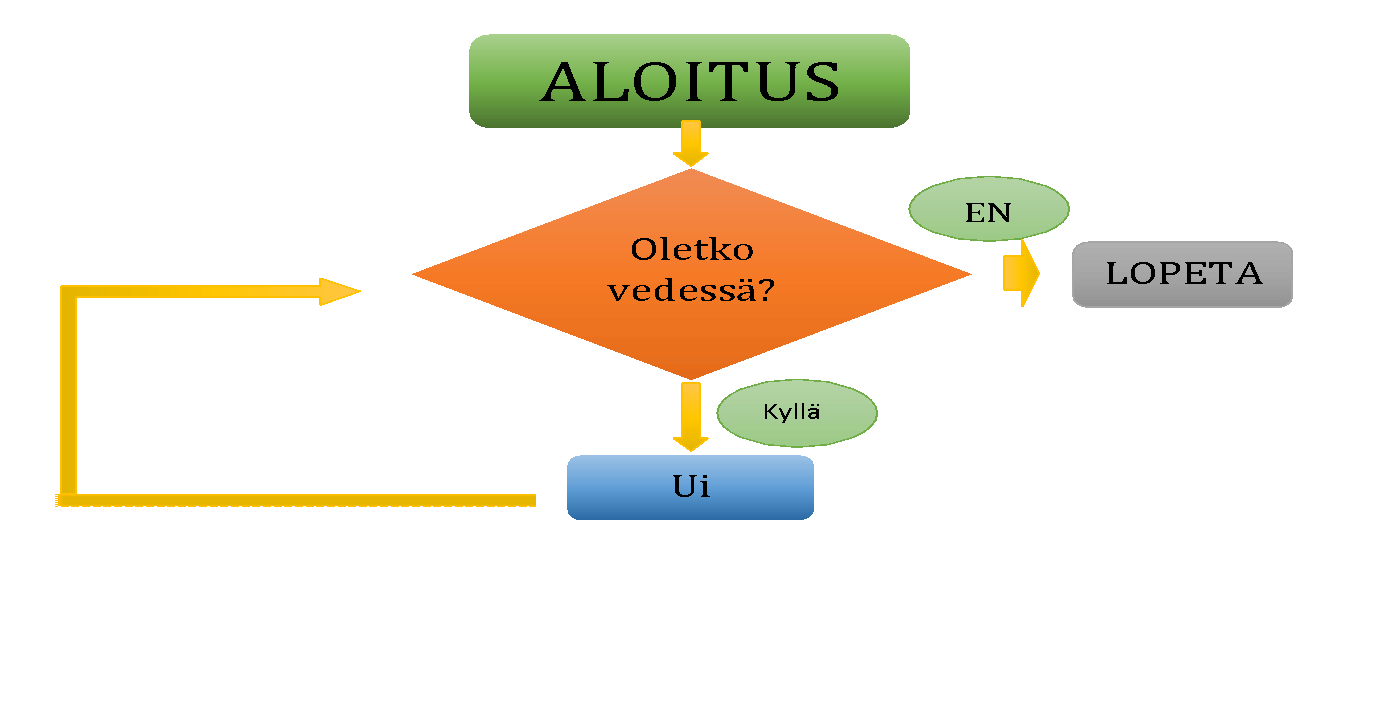
Toisto, jota kutsutaan usein myös iteraatioksi tai silmukaksi, on rakenne, jonka avulla käsky (tai joukko käskyjä) voidaan suorittaa useammin kuin kerran ilman, että niitä tarvitsee toistaa joka kerta peräkkäisen käskylohkon yhteydessä. Toistoja on yleensä kolmea eri tyyppiä: *count-controlled* (kutsutaan myös for-silmukaksi, koska eri ohjelmointikielissä käytetään eri avainsanoja, joilla viitataan tämäntyyppiseen toistoon), *while* ja *repeat-until*.

* 1. **“Count-controlled” -toisto**

Count-controlled -toisto on toistotyyppi, joka sallii käskylohkon toistamisen ennalta määritetyn määrän kertoja. Jos esimerkiksi haluaisimme ohjeistaa, miten laskea alaspäin 10:stä 0:aan, voisimme sanoa: "vähennä 1 nykyisestä lukemasta 10 kertaa peräkkäin". Tämä rakenne on hyödyllinen sellaisten operaatioiden yhteydessä, joista tiedämme jo etukäteen, että ne toistetaan tietyn määrän kertoja ja joiden toistaminen ei riipu mistään muusta ehdosta.

* 1. **“While”-toisto**

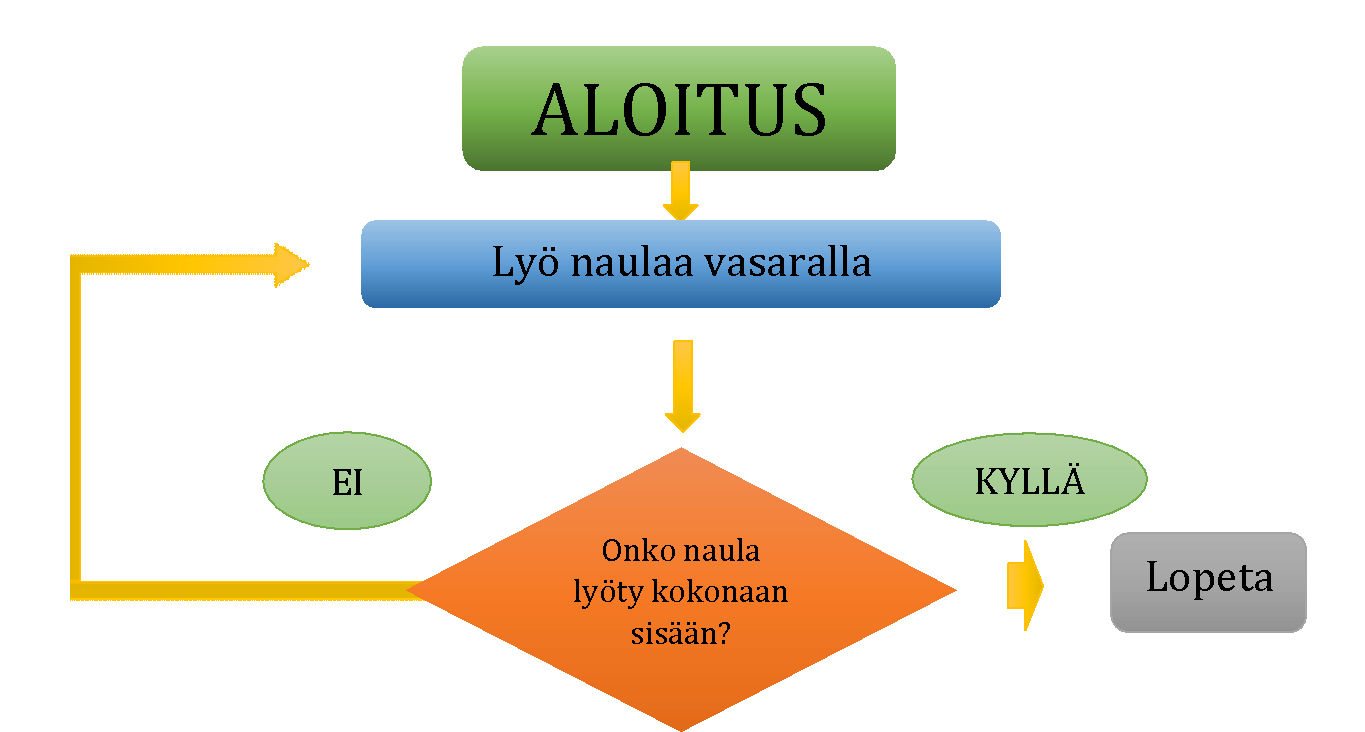
Toisin kuin count-controlled, while-toisto mahdollistaa käskykokonaisuuden toistamisen, kunnes tietty ehto täyttyy. Tämä rakenne mahdollistaa sen, että ohjeet voidaan määritellä suoritettavaksi useita kertoja tietämättä kuitenkaan etukäteen, kuinka monta kertaa ne toistetaan. Esimerkki tämäntyyppisestä suorituksesta on seuraava käsky: "kun olet vedessä, ui".



On tärkeää huomata, että while-toistoa käytettäessä meillä ei ole varmuutta siitä, että toistettavaksi tarkoitettu käskylohko suoritetaan vähintään kerran. Tässä esimerkissä, jos suorittaja on alusta alkaen poissa vedestä, käskyä "uida" ei koskaan suoriteta.

* 1. **“Repeat until” -toisto**

Repeat until -toisto on itse asiassa hyvin samanlainen rakenne kuin *while*: sitäkin käytetään, kun käskykokonaisuus on mahdollisesti toistettava useita kertoja, emmekä tiedä etukäteen, kuinka monta kertaa se on toistettava. Ainoa ero on se, että pysäytysehdon tarkistus suoritetaan käskyn ensimmäisen iteraation jälkeen. Toisin sanoen varmistetaan, että ohjeet suoritetaan vähintään kerran ennen kuin päätetään, toistetaanko ne vai ei.



# Pseudokoodi

Vuokaaviot ovat erinomainen väline algoritmin formalisoimiseksi ymmärrettävällä ja kirjoitettavalla kielellä. Niin hyödyllisiä kuin ne ovatkin, niillä on kuitenkin merkittäviä haittoja, jotka tekevät niiden käytöstä usein hankalaa: suuri tilantarve ja tyypillisten toistorakenteiden (esim. count-controlled) vaikea kartoittaminen ja kuvaaminen. Suosittu vaihtoehto onkin pseudokoodi: kieli, joka muistuttaa ohjelmointikieliä, mutta abstrahoi niistä tiettyjä avainsanoja, pysyy yleismaailmallisena ja lähempänä luonnollista kieltä, jota käytämme päivittäin.

Tässä luvussa esittelemme lyhyesti yksinkertaistetun pseudokoodin, jonka avulla voimme helposti kirjoittaa omia algoritmejamme käyttämällä esitettyjä rakenteita. Tämän kielen sisällä käytämme seuraavia sääntöjä:

* jokainen rivi edustaa ohjetta

| tämä on yksi ohje  tämä on toinen ohje  tämä on toinen ohje |
| --- |

* Ohjelohkot on esitettävä käyttämällä sisennystä: eri lohkoilla on eri määrä tyhjiä välejä ennen riviä.

| tämä ohje kuuluu **lohkoon A**  tämäkin käsky kuuluu **lohkoon A**  tämä käsky kuuluu lohkoon B  tämä ohje kuuluu myös lohkoon B  tämä ohje kuuluu myös lohkoon B  tämä ohje kuuluu **lohkoon A** |
| --- |

* Valinta tunnistetaan avainsanalla **IF**, jota seuraa tarkistettava ehto ja sitten uusi käskylohko, joka suoritetaan, jos ehto on tosi. Tämän lohkon jälkeen voidaan käyttää toista käskylohkoa, jolla määritetään käsky, joka suoritetaan, jos ehto on epätosi; tämä lohko on esitettävä avainsanalla **ELSE**.

| ohje  toinen ohje  **IF-ehto**  **tämä käsky suoritetaan, jos ehto on tosi.**  **Tämä käsky suoritetaan myös, jos ehto on tosi.**  **ELSE**  **tämä käsky suoritetaan, jos ehto on väärä.**  **Tämä käsky suoritetaan myös, jos ehto on väärä.**  tämä käsky suoritetaan IF:n jälkeen määritellystä ehdosta riippumatta |
| --- |

* Count controlled -toisto tunnistetaan avainsanalla **FOR X TIMES DO**, jota seuraa X kertaa toistettava käskylohko.

| ohje  muu ohje  **FOR X TIMES DO**  **tämä ohje toistetaan X kertaa**  **tämäkin käsky toistetaan X kertaa**  tätä ohjetta ei toisteta |
| --- |

* Toiston aikana toistaminen tunnistetaan avainsanalla **WHILE <CONDITION>** **DO**, jota seuraa käskylohko, jota toistetaan, kunnes <CONDITION> ei enää ole totta. Muista, että jos ehto ei aluksi ole tosi, seuraavan lohkon ohjeita ei suoriteta.

| ohje  muu ohje  **WHILE CONDITION DO**  **tämä käsky toistetaan, kunnes ehto pitää paikkansa**  **tämäkin käsky toistetaan, kunnes ehto pitää paikkansa**  tätä ohjetta ei toisteta |
| --- |

* Repeat until -toisto tunnistetaan avainsanalla **REPEAT**, jota seuraa käskylohko. Tätä käskylohkoa on seurattava avainsana **UNTIL** <**CONDITION**>, joka määrittää tarkistettavan ehdon, jotta voidaan määritellä, toistetaanko käskylohko vai ei.

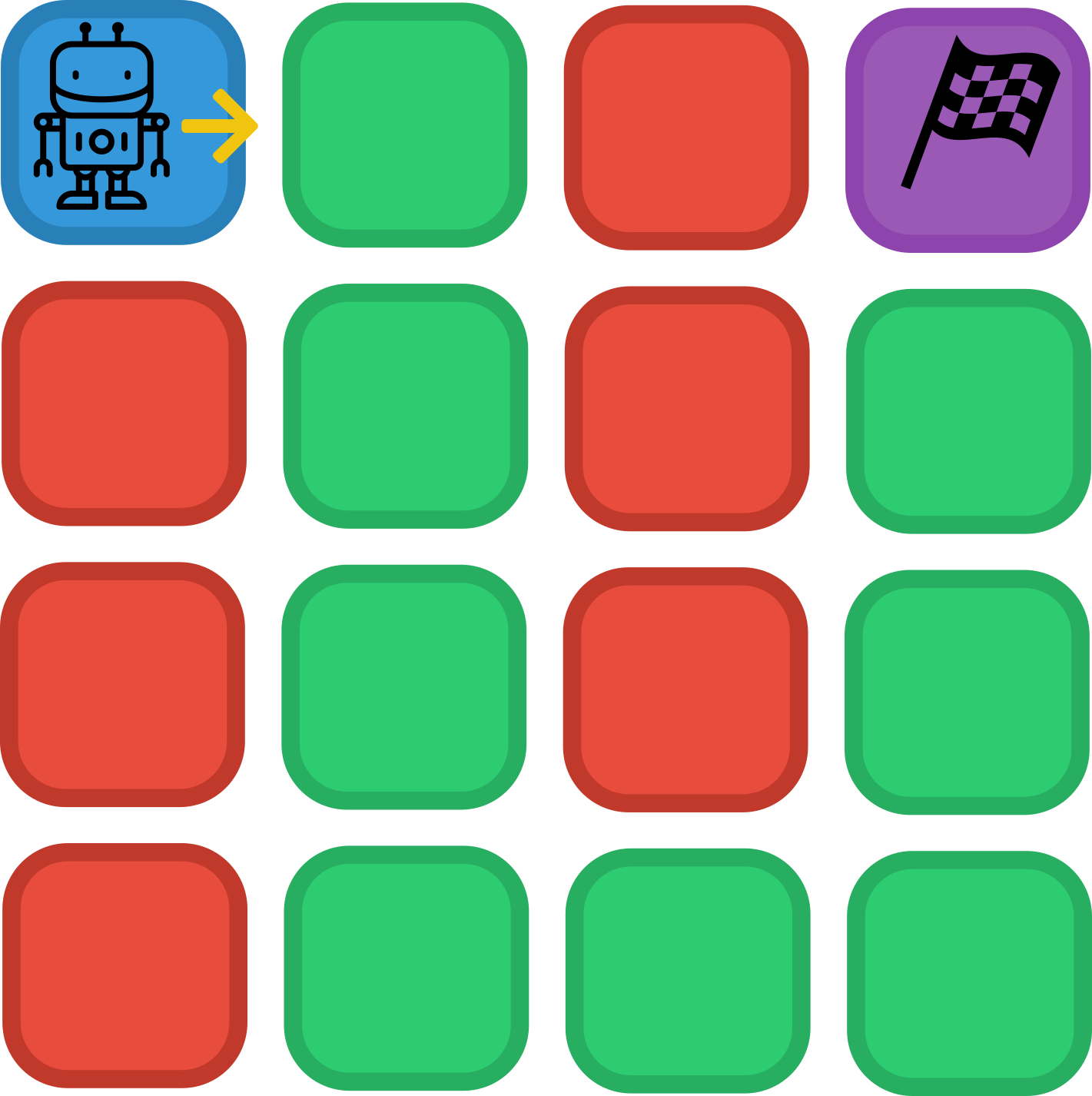
| ohje  muu ohje  **REPEAT**  **tämä käsky toistetaan, kunnes ehto pitää paikkansa**  **tämäkin käsky toistetaan, kunnes ehto pitää paikkansa**  **UNTIL CONDITION**  tätä ohjetta ei toisteta |
| --- |

# Esimerkkitehtävät

1) Bob on pieni robotti. Koska se on hyvin yksinkertainen, sillä on vain kolme toimintoa:

* käänny vasemmalle 90°
* käänny oikealle 90°
* kävele yksi laatta eteenpäin

Osaatko auttaa Bobia pääsemään siniseltä laatalta violetille laatalle seuraavan kuvan mukaisesti liikkumalla vain vihreiden laattojen kautta? Kirjoita kaksi algoritmia tätä tehtävää varten, toinen ilman ja toinen toistokonstruktioita käyttäen. Huomaa, että keltainen nuoli osoittaa, mihin suuntaan Bob lähtee katsomaan.



| **Ratkaisu ilman toistoa** | **Ratkaisu toistorakenteiden avulla** |
| --- | --- |
| kävele yksi laatta eteenpäin  käänny oikealle 90°  Kävele yksi laatta eteenpäin  Kävele yksi laatta eteenpäin  Kävele yksi laatta eteenpäin  käänny vasemmalle 90°  Kävele yksi laatta eteenpäin  Kävele yksi laatta eteenpäin  käänny vasemmalle 90°  Kävele yksi laatta eteenpäin  Kävele yksi laatta eteenpäin  Kävele yksi laatta eteenpäin | kävele yksi laatta eteenpäin  käänny oikealle 90°  FOR 3 TIMES DO  kävele yksi laatta eteenpäin  käänny vasemmalle 90°  FOR 2 TIMES DO  kävele yksi laatta eteenpäin  käänny vasemmalle 90°  FOR 3 DO  kävele yksi laatta eteenpäin |

# Viitteet

[1] Böhm, C., & Jacopini, G. (1966). Flow diagrams, turing machines and languages with only two formation rules. *Commun. ACM, 9*, 366-371.

1. Dooderin luoma ja Flaticonista ladattu mokkapannun kuvake - "https://www.flaticon.com/free-icons/moka-pot". [↑](#footnote-ref-0)