Ohjelmoinnillinen ajattelu ja ohjelmointiympäristöt

Joustavaan matematiikkaan -täydennyskoulutushanke

Materiaalin kuvaus: Tässä tekstissä esitellään yksi tulkinta yläkoulun matematiikan opetussuunnitelman ohjelmoinnin tavoitteesta: “T20 ohjata oppilasta kehittämään algoritmista ajatteluaan sekä taitojaan soveltaa matematiikkaa ja ohjelmointia ongelmien ratkaisemiseen”. Tulkinta on toiminut taustateoriana Ohjelmoinnillinen ajattelu ja ohjelmointiympäristöt -kurssin yläkoulun ohjelmoinnin opetuksen sisällölle. Lukion opettajille kurssi antoi käsityksen ohjelmoinnin perusteista. Täydennyskoulutuksen teeman mukaisesti ohjelmoinnin tavoitteen tulkinnassa on hyödynnetty joustavan ajattelun näkökulmaa.

Materiaali on tuotettu osana Joustavaan matematiikkaan -hanketta (JoMa). JoMa on vuosina 2018–2023 toiminut valtakunnallinen matematiikan opetuksen täydennyskoulutushanke varhaiskasvatukseen, esiopetukseen, alakouluun, yläkouluun ja lukioon. Hankkeessa tuotettiin 19 verkkokurssia. Kursseilla kehitettyä materiaalia löytyy täältä Avointen oppimateriaalien kirjastosta. Opetushallituksen rahoittaman hankkeen toteuttamiseen osallistuivat Turun yliopisto, Åbo Akademi, Jyväskylän yliopisto ja Oulun yliopisto.

Materiaalin tekijät: Peter Larsson, Marika Parviainen ja Heidi Kaarto

Lisenssi: Tämä teos on lisensoitu Creative Commons Nimeä 4.0 Kansainvälinen -käyttöluvalla. Tarkastele käyttölupaa osoitteessa https://creativecommons.org/licenses/by/4.0/deed.fi.

ALGORITMINEN AJATTELU, OHJELMOINTI JA ONGELMANRATKAISU MATEMATIIKASSA

Joustavaan Matematiikkaan
www.flexibility.fi

Sisältö

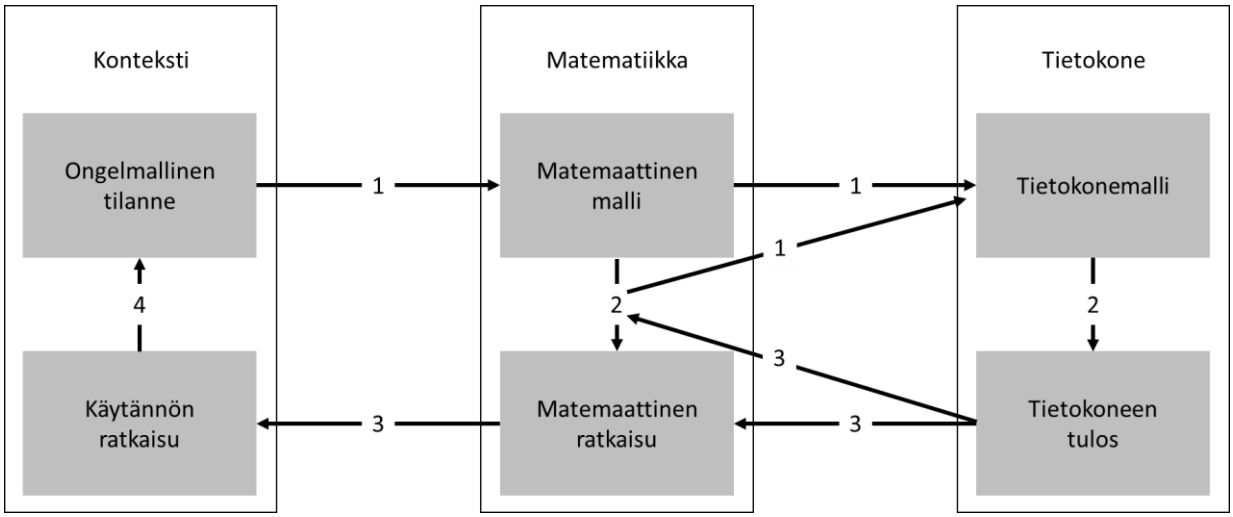
1 Johdanto 1
2 Ongelmanratkaisu ja algoritminen ajattelu matematiikassa 3
3 Joustava matematiikka ja algoritmit 9
4 Matemaattisesta ongelmanratkaisusta tietokonemalliin 10
5 Ongelmanratkaisu ja algoritminen ajattelu ohjelmoinnissa 12
6 Joustava matematiikka ja ohjelmointi 16
7 Algoritminen ajattelu ja ongelmanratkaisu 17
Lähteet 18

1 Johdanto

Matematiikan ja ohjelmoinnin yhteys juontaa ensimmäisestä tietokoneesta, joka luotiin elektroniseksi laskimeksi. Poiketen sen ajan mekaanisista laskimista tietokone pystyi tallentamaan välitulokset ja hyödyntämään niitä uusissa laskutoimituksissa. Ensimmäisen tietokoneen ohjaaminen oli kuitenkin konekeskeistä, jossa koneen toiminta määriteltiin kytkimillä ja johdotuksilla. Vuonna 1945 julkaistussa nykyaikaisen tietokoneen mallissa (Neumann 1945) toimintaperiaate oli muuttunut matemaattiseksi ja ohjaaminen tapahtui symbolien avulla. Toimintaperiaatteella tarkoitetaan peräkkäis-, ehto- ja toistorakenteita, joita käytetään koneen ominaisuuksien suoritusjärjestyksen ohjaamiseen. Ohjausrakenteita voidaan pitää matemaattisina, koska niillä voidaan myös kuvata laskutoimitusten algoritmien etenemistä.

Tietokoneessa algoritmi ohjaa koneen toimintaa, joten sitä voidaan tulkita sekä matemaattisesta että teknisestä näkökulmasta. Ensimmäisestä tulkinnasta vastaa peruskoulun opetuksessa matematiikka ja toisesta käsityö. Keskitymme tässä matematiikkaan, mutta on hyvä huomioida myös käsityön näkökulma, koska osana matematiikka käytämme myös tietokoneen teknisiä ominaisuuksia.

Yläkoulun matematiikan opetussuunnitelma määrittelee ohjelmoinnin keskeiseksi tavoitteeksi (T20) "ohjata oppilasta kehittämään algoritmista ajatteluaan sekä taitojaan soveltaa matematiikkaa ja ohjelmointia ongelmien ratkaisemiseen" (OPH, s.375, 2014). Ratkaisemisella tarkoitetaan tässä tosielämän ongelman kuvaamista, kuvauksen muuttamisesta matemaattiseksi malliksi (kaava) ja ratkaisun laskemista (Kallia ym. 2021; ks. Kuvio 1.1). Tapauksessa, jossa tarvitaan tietokoneen ominaisuuksia, matematisoitu ongelma voidaan mallintaa ohjelmointikielellä ja suorittaa malli tietokoneella.



Kuvio 1.1 Matemaattista ja tietokoneen ohjelmointiin perustuvaa ongelmanratkaisua yhdistää algoritmin laskeminen (2). Lähtökohtana on ongelman kuvaus, jonka sisältämistä käsitteistä ja niiden välisistä suhteista luodaan (1) ongelmaa kuvaava kaava eli ongelman matemaattinen malli. Kaavan voi ratkaista laskemalla käsin (2), mutta jos laskeminen on hankalaa tai dataa on paljon, voidaan käyttää tietokonetta apuvälineenä. Silloin luodaan tietokonemalli kaavasta ohjelmoimalla (1), jonka perusteella tietokone laskee (2) tuloksen. Tietokoneen tulosta voi hyödyntää (3) sekä kokonaisena että osittaisena matemaattisena ratkaisuna. Mahdollisen matemaattisen käsittelyn jälkeen pitää ratkaisu vielä tulkita ongelman kontekstissa (3), ennen kuin ongelma on ratkaistu käytännössä (4).

Ongelman kuvaamisella tarkoitetaan sen keskeisten piirteiden esille tuontia. Kuvauksen matemaattisella analyysilla pyritään esittämään piirteet täsmällisesti vakioina, muuttujina ja niiden välisinä suhteina. Lopputuloksena on ongelman matemaattinen malli eli sitä kuvaava kaava, joka voidaan ratkaista laskemalla. Laskutoimitus on algoritmin noudattamista, jonka tulos on tuotava takaisin ongelman kontekstiin sen ratkaisemiseksi. Ongelman kontekstin huomioiminen vastauksessa on tärkeää, koska siinä voi olla reunaehtoja, jotka eivät ole matemaattisia (Rich & Yadav 2020). Esimerkiksi tosielämässä ei tilata osittaisia busseja vaan kokonaisia.

Ohjelmoimalla voidaan luoda matemaattisista malleista tietokonemalleja eli ohjelmia, kun laskeminen olisi muuten liian työlästä, tarvitaan nopeita kokeiluja tai ratkaisu saadaan usean laskutoimituksen vuorovaikutuksesta nk. simulaatiot. Ohjelman suorituksen tuloksesta palataan matemaattiseen kontekstiin, koska tulos voi olla laskutoimituksen osa tai tulos voi vielä vaatia matemaattista tarkastelua.

Tässä tekstissä keskitymme ongelmanratkaisuun luomalla matemaattisia ja tietokonepohjaisia malleja. Ratkaisu perustuu molemmissa algoritmiseen ajatteluun, mutta samalla huomataan, että matematiikassa ja ohjelmoinnissa algoritmia tulkitaan eri tavalla.

2 Ongelmanratkaisu ja algoritminen ajattelu matematiikassa

Yksinkertaiset laskutoimitukset perustuvat pitkälti ulkoa opittuun ja aina ei välttämättä edes ajatella laskemista vaan vastaus tiedetään. Kun laskeminen menee monimutkaisemmaksi, niin se muuttuu symbolien manipuloimiseksi ja usein käytetään lisäksi ulkoisia apuvälineitä. Symbolien manipulointi on algoritmista alkaen allekkainlaskuista ja jatkuen algebran kaavojen käsittelyyn. Myös harpin ja viivaimen käyttö geometriassa on esimerkki algoritmista. Ihmisen suorittamana algoritmi muodostuu niistä proseduureista, joita tarvitaan laskutoimituksen tuloksen saamiseksi. Proseduurien valinta tapahtuu niiden oppimisen jälkeen monesti tiedostamatta.

Ongelmanratkaisu on yksinkertaisimmillaan ongelmaan sopivan algoritmin hakemista ja suorittamista. Aina ongelmat eivät kuitenkaan ole näin suoraviivaisia. Algoritmin taustalla olevien matematiikan käsitteiden ymmärtäminen mahdollistaa ratkaisun perustamisen ongelman luonteeseen. Silloin kiinnitetään huomiota ongelmaa kuvaaviin käsitteisiin ja niiden välisiin suhteisiin, joiden perusteella valitaan tai päätellään ratkaisun kaava. Usein ongelma voidaan ratkaista usealla tavalla, jolloin on pystyttävä arvioimaan, mikä on kyseisessä tilanteessa paras. Välillä ongelmaan ei kuitenkaan löydy valmista algoritmia vaan sellainen on luotava. Nämä kaikki ovat esimerkkejä algoritmisesta ajattelusta, joka voidaan jakaa kolmeen osaamisen tasoon: tieto ja taito, käsitteellinen ymmärtäminen sekä arviointi ja luominen (Fan & Bokhove 2014; ks Taulukko 2.1).

Taulukko 2.1 Algoritmisen ajattelun tasot.

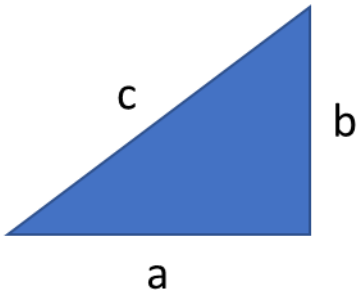
Taso	Kuvaus
1. Tieto ja taito	Tuntee kaavan sisältämät proseduurit, jonka perusteella osaa suorittaa laskutoimituksen. Soveltaminen on tällä tasolla suoraviivaista ja algoritmin muokkaaminen on vähäistä.
2. Käsitteellinen ymmärtäminen	Tietää miksi algoritmi toimii ja osaa soveltaa sitä erilaisiin ongelmiin.
3. Arviointi ja luominen	Pystyy arvioimaan erilaisten algoritmien paremmuutta (tehokkuus tai yksinkertaisuus). Osaa yleistää algoritmin sopimaan samantyyppisiin ongelmiin ja luoda uusia algoritmeja.

Tieto ja taito

Tiedon ja taidon tasolla osaaminen liittyy laskukaavan ja sen toteuttavan algoritmin muistamiseen. Oppilas taitaa tarvittavat proseduurit ja pystyy yhdistämään ne kaavan määrittelemän laskutoimituksen suorittamiseksi. Tällä tasolla osaaminen on mekaanista ja se soveltuu hyvin suoraviivaisiin laskutehtäviin. Opetusmuodoksi riittää tiedon siirtäminen, jossa edetään opettajajohtoisesti. Opettaja kertoo, mihin kaavaa on tarkoitus käyttää ja esittää malliksi siihen perustuvan laskutoimituksen, minkä jälkeen oppilaat harjoittelevat joukolla tehtäviä. Tehtävän tarkastuksessa etsitään virheitä, jotka korjataan saattamalla laskutoimitus ratkaisumallin mukaiseksi. Oppilas voi ratkaista oppimisen opettelemalla kaavan ja tarvittavat proseduurit ulkoa. Esimerkkeinä algoritmeista ovat alakoulusta tuttu kertolasku allekkain aritmetiikassa ja yläkoulussa Pythagoraan lause geometriassa (ks. Kuvio 2.1).

			7	0	
		*	6	4	
		2	8	0	2
+	4	2	0		4
	4	4	8	0	

$$a^2 + b^2 = c^2$$
$$c = \sqrt{a^2 + b^2}$$
$$5 = \sqrt{4^2 + 3^2}$$



Kuvio 2.1 kertolasku allekkain ja Pythagoraan lause sekä toteutus.

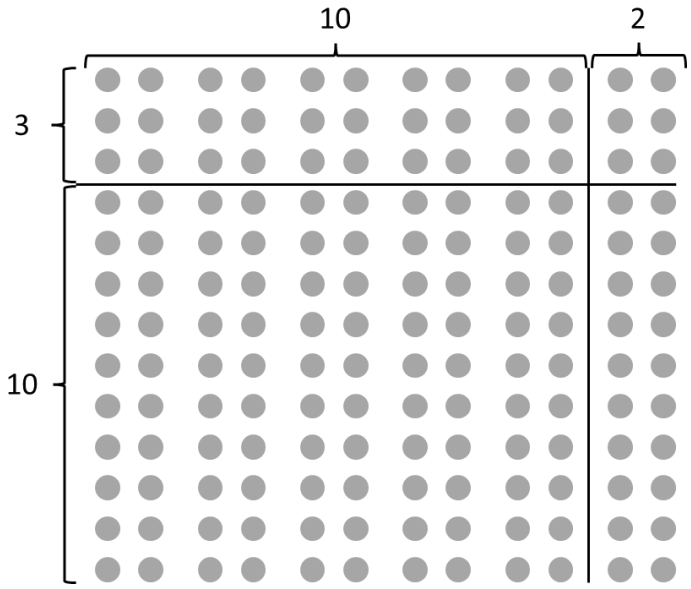
Yllä oleva kuvio kertoo miten kertolasku ja suorakulmaisen kolmion sivun pituus voidaan laskea. Ensiksi mainitussa algoritmina on allekkainlasku ja jälkimmäisessä Pythagoraan lause. Oppilas pystyy näillä ratkaisemaan ongelmat, jotka viittaavat juuri näiden menetelmien askeliin. Kun ongelmat ovat monimutkaisempia tai vaativat menetelmien soveltamista, pelkän ratkaisutavan osaaminen ei riitä, vaan tarvitaan sen lähtökohtien ymmärtämistä.

Käsitteellinen ymmärtäminen

Käsitteellisen ymmärtämisen tasolla osaamisella tarkoitetaan yksittäisten algoritmien ymmärtämistä, johon kuuluu algoritmin matemaattisten perusteiden tunteminen ja kyky soveltaa algoritmia monimutkaisiinkin ongelmiin. Tällä tasolla osaaminen perustuu käsitteelliseen tietoon, joka sisältää matematiikan osa-alueen käsitteet ja niiden väliset suhteet. Oppilas ymmärtää miten algoritmi on seurausta matematiikan osa-alueen rakenteesta. Opetusmuoto on konstruktiivinen. Opettaja kertoo mitä algoritmi merkitsee ja minkä takia se toimii. Hän perustelee, miten algoritmin voi päätellä osa-alueen säännöistä ja käsitteistä. Opettaja tukee oppilaiden tiedon luomista liittämällä opetettavat asiat heidän olemassa olevaan tietoon.

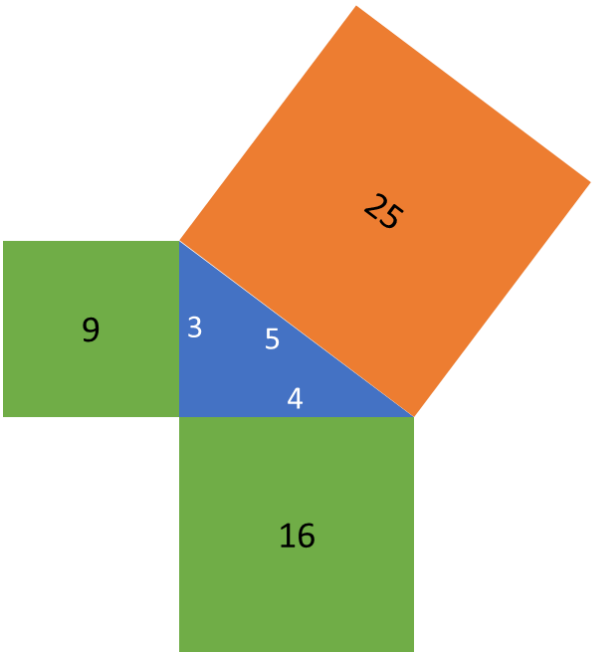
Pelkkä symbolikieli ja sen manipulointi ei välttämättä riitä tuomaan esille matematiikan osa-alueen rakennetta vaan tarvitaan lisäksi esim. visuaalista tai sanallista selitystä. Tätä kutsutaan kielentämiseksi, johon kuuluu matematiikan symbolikieli, luonnollinen kieli, kuviokieli ja taktiilinen toiminnan kieli (Joutsenlahti & Tossavainen 2018). Symbolinen kieli on ns. perinteinen tapa esittää matematiikka, joten kielentämisessä hyödynnetään lisäksi yhtä tai useampaa muuta esitystapaa.

Moninumeroisilla luvuilla kertomisen visualisointi, on esimerkkki kielentämisestä. Kertominen voidaan esittää alkioista koostuvana taulukkona, josta toinen luku määrittelee kuinka monta alkiota on rivissä ja toinen rivien määrän (ks. Kuvio 2.2). Lopputulos saadaan laskemalla alkioiden summa, joka on kertolaskun tulos. Tuloksen ymmärtää visuaalisesti, mutta samalla siitä käy esille aritmetiikan käsitys määristä, lukujen suhteesta niihin ja kertolaskusta.



Kuvio 2.2 kertolasku 12 kertaa 13 on esitetty visuaalisesti: siinä näkyvät lukujen edustamat määrät ja kertolaskun vaikutus tulokseen.

Toinen esimerkki laskutoimituksen kielentämisestä on Pythagoraan lauseen visualisointi (ks. Kuvio 2.3). Voimme esittää lauseen suorakulmaisena kolmiona, joiden sivuille on piirretty neliöt. Kun neliöiden koot mitataan, huomataan, että hypotenuusasta muodostuva neliö, on kateettien neliöiden summa. Hypotenuusan pituus saadaan laskemalla kateettien neliöiden summan neliöjuuri.



Pythagoraan lause:

$$a^2 + b^2 = c$$
$$4^2 + 3^2 = 5^2$$
$$16 + 9 = 25$$

Hypotenuusan pituus:

$$c^2 = a^2 + b^2$$
$$c = \sqrt{a^2 + b^2}$$
$$c = \sqrt{16 + 9} = \sqrt{25} = 5$$

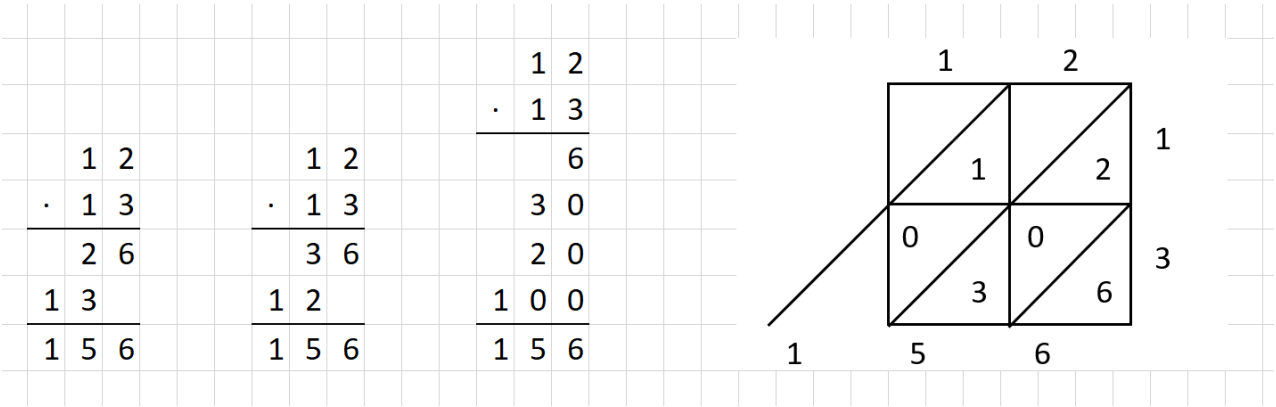
Kuvio 2.3 Pythagoraan lauseen visualisointi.

Visualisointi mahdollistaa laskutoimituksen perustamisen matematiikan osa-alueen käsitteelliseen rakenteeseen. Laskutoimituksen ymmärtämiseksi ei kuitenkaan riitä, että visualisointi esitetään laskukaavan yhteydessä. Myös visualisointia tulisi opettaa kuten laskemistakin, jotta siitä tulee osa oppilaiden matemaattista taitoa. Vastaavasti oppilaiden tulisi harjoitella myös muita kielentämisen muotoja.

Laskutoimituksen perusteiden ymmärtäminen auttaa menetelmän soveltamisessa. Usein on kuitenkin mahdollista ratkaista sama ongelma tai laskea kaava usealla algoritmilla, jolloin on hyvä tietää miten valitaan sopivin. Monet ongelmat ovat samantyyppisiä, jolloin on osattava yleistää tietty ratkaisu koskemaan samanlaisia ongelmia. Matematiikan ulkopuoliset ongelmat voivat lisäksi vaatia useamman algoritmin, jolloin on osattava yhdistää eri kaavoja. Nämä kysymykset vaativat korkeamman tason osaamista, jossa oppilas pystyy arvioimaan ja luomaan algoritmeja.

Arviointi ja luominen

Arvioinnin ja luomisen tasolla osaaminen ei koske enää yksittäisten algoritmien tuntemista vaan silloin pystytään vertaamaan erilaisten algoritmien paremmuutta, havaitsemaan miten tietystä ratkaisusta voisi luoda yleiskäyttöisemmän algoritmin ja luomaan uusia algoritmeja aikaisemmin opittuja yhdistelemällä. Tavallisistakin laskutoimituksista löytyy erilaisia algoritmeja (Morgan 2019; ks. Kuvio 2.4). Valinta voi tapahtua sen mukaan minkä algoritmin on oppinut, mutta perustellumpia kriteereitä ovat yksinkertaisuus ja tehokkuus. Yksinkertaisuuden voidaan ajatella liittyvän algoritmin askelten suorittamisen vaikeusasteeseen, kun taas tehokas menetelmä minimoi askelten määrän.



Kuvio 2.4 Tavallinen kertolasku esim. 12·13 voidaan laskea usealla eri tavalla. Kolmessa ensimmäisessä laskutavassa (vasemmalta), on laskettu kertolasku allekkain. Ensimmäisessä laskutavassa kertojana toimii luku 13, mutta toisessa 12. Kolmannessa luku 13 on kertojana, mutta kertolaskun tulokset lasketaan numero kerrallaan niin, että kahden numeron kertolaskun tulos tulee omalle rivilleen huomioiden paikan kymmenjärjestelmässä. Lopuksi lasketaan kaikki yhteen. Viimeisessä tavassa käytetään kertolaskussa apuna kehikkoa, jossa kerrottava on neliön yläpuolella (12) ja kertoja oikealla sivulla (13). Kerrottavan ja kertojan numeroiden risteyskohdat kuvataan neliöinä, jotka edustavat kahden numeron kertolaskua. Neliöt on jaettu vinoittain oikeasta ylänurkasta vasempaan alanurkkaan. Alaosassa on kertolaskun tulos ja yläosassa on mahdollinen muistiluku. Tulos lasketaan vinottain oikealta vasemmalle ja ylhäältä alas, jolloin summataan samassa linjassa olevien neliöiden puolikkaat. Silloin saadaan automaattisesti mahdolliset muistiluvut mukaan kertolaskun tulosten yhteenlaskuun paikkajärjestelmän mukaisesti. Kertolaskujen ja muistilukujen tulokset kirjataan jokaisen paikkajärjestelmän linjan mukaisesti neliöiden muodostaman kehikon alle.

Koulussa opitut algoritmit ovat yleiskäyttöisiä eli ne sopivat samantyyppisten ongelmien ratkaisuun. Kun sovellamme laskukaavaa tiettyyn ongelmaan, voimme huomata, että jättämällä jotkut arvot avoimeksi tai tietyn arvoalueen sisään voimme käyttää kaavaa joukkoon samantyyppisiä ongelmia. Matemaattisemmin tämä liittyy algebran osa-alueeseen, jossa voidaan esittää laskutoimituksen tekijöitä muuttujista ja vakioista koostuvina kaavoina. Erilaiset operaattorit kertovat muuttujien ja vakioiden välisistä suhteista. Peruskoulun matematiikassa erilaiset sovelluskohteet matematiikan ulkopuolelta voivat tarjota algoritmin yleistämisen mahdollisuuksia kuten ostaminen, rakentaminen, urheilu, harrastukset, jne.

Algoritmin ja sitä edustavan kaavan yleistäminen on uuden luomista. Se laajentaa konstruktivistista oppimista vaatimalla myös kekseliäisyyttä ja luovuutta. Joskus tarvittava algoritmi ei ole itsestään

selvä, jolloin se muodostaa ongelman, jonka ratkaiseminen ei välttämättä heti onnistu. Algoritmin luominen voi vaatia useamman yrityksen ja erehdyksen, kunnes löytyy ongelman kaikki piirteet huomioiva ratkaisu. Ongelma voidaan pilkkoa pienemmiksi ongelmiksi helpottamaan ratkaisun löytämistä. Joskus aluksi useamman askeleen vaatima algoritmi voidaan tiivistää kaavaksi, jossa muutamalla laskutoimituksella (askeleella) saadaan tulos (ks. Kuvio 2.5).

Suklaalevyä halkoo pysty- ja vaakasuuntaiset uurteet jotka muodostavat kymmenen palaa. Kuinka monella leikkauksella suklaalevy voidaan jakaa palasiin, jos jokainen leikkaus aina jakaa levyn kahteen osaan?

Voit helposti laskea askeleet ratkaisuun, mutta **pyri löytämään invariantti**, joka on voimassa lähtöasetelmasta loppuasetelmaan.

Vastaus: 9 leikkausta kun paloja on 10. Paloja (n) on aina yksi enemmän kuin leikkauksia (invariantti), joten leikkausten määrän saa kaavalla $n - 1$.

Kuvio 2.5 tehtävässä haetaan leikkausten määrää laskevaa kaavaa, jolla suklaalevyn saa jaettua sen kuvion mukaisiin osiin.

Algoritmin arviointi ja luominen vaatii sekä kriittistä että luovaa ajattelua. Arviointia varten on sekä tunnettava monenlaisia algoritmeja että ymmärrettävä, miten ne liittyvät taustalla oleviin käsitteisiin. Kuitenkin arviointi on myös laadullista: on pystyttävä luomaan kriteerit, joiden perusteella yksi algoritmi on toista parempi. Uuden algoritmin luominen, joko muokkaamalla erityiseen ongelmaan tarkoitetusta algoritmista samantyyppisiin ongelmiin soveltuva ratkaisu tai keksimällä täysin uusi (ks. edellä 2.5), voidaan sanoa olevan algoritmisen ajattelun huipentuma. Opettamisen kannalta arvioinnin harjoittelu on suoraviivaisempaa, mutta myös kekseliäisyyttä voidaan harjoitella.

On epätodennäköistä, että oppilaat keksivät matematiikan osa-alueisiin liittyviä uusia algoritmeja, mutta myös uudelleen keksiminen on opettavaa. Matematiikan ulkopuolisten ongelmien ratkaisu sen sijaan luo mahdollisuuden keksiä omia algoritmeja. Ohjatussa oppimisessa opettaja antaa vihjeitä ja tarjoaa tukirakenteita algoritmien keksimiseen. Vapaamuotoisessa oppimisessa oppilas luo ongelman ratkaisevan algoritmin itsenäisesti. On tärkeää, että kekseliäisyyttä harjoitellaan ensin ohjatusti ja sitten vapaamuotoisemmin. Toimiminen ryhmässä voi antaa tukea vapaamuotoisessa ongelmanratkaisussa. Kun ongelma on matematiikan ulkopuolelta, on sen kuvaus ensin muutettava

matemaattiseksi eli mallinnettava. Mallinnettaessa luodaan ongelmaa kuvaava matemaattinen kaava, joka on lähtökohta ratkaisun tarjoavan algoritmin suorittamiselle.

3 Joustava matematiikka ja algoritmit

Joustava matemaattinen ajattelu tarkoittaa sitä, että ei seurata vain yhtä toimintamallia vaan tilanteesta riippuen hyödynnetään erilaisia ajattelutapoja. Joustava ajattelija on valmis mukauttamaan ja vaihtamaan ajattelutapaansa tehtävään sopivaksi.*

Esimerkkejä joustavasta ajattelusta*

- Ratkaisee ongelman/tehtävän monella eri tavalla
- Valitsee tehokkaan ratkaisutavan
- Vaihtaa ratkaisutavasta toiseen
- Yhdistelee ratkaisutapoja
- Mukauttaa ratkaisutavan uuteen tilanteeseen sopivaksi
- Muodostaa uuden ratkaisutavan aiempien tietojen pohjalta
- Tulkitsee/analysoi erilaisia ratkaisutapoja

Joustavuuden voi ajatella olevan jatkumo, jossa vähitellen pystyy yhä syvällisempään joustavuuteen. Lopullinen tavoite on, että pystyy uudessa ongelmanratkaisutilanteessa arvioimaan erilaisia (ennalta tuntemattomiakin) lähestymistapoja ja jopa kehittämään uuden tilanteeseen sopivan lähestymistavan.*

Matemaattisesta kontekstista riippuen joustavuudella on erilaisia ilmenemismuotoja. Esimerkiksi seuraavat*:

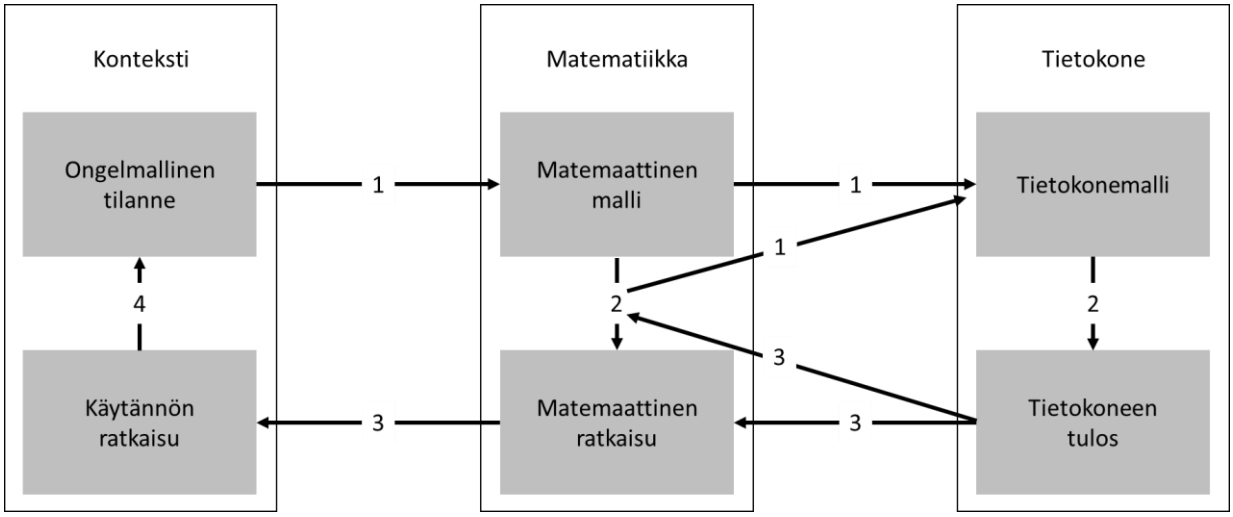
- Joustavuus ratkaisustrategian valinnassa
- Joustavuus esitysmuotojen käytössä

Joustava matemaattinen ajattelu menetelmän tai ratkaisustrategian valinnassa liittyy aiemmin tässä osiossa esitettyihin kolmeen algoritmisen ajattelun tasoon. Tiedon ja taidon -tasolla osaaminen on mekaanista ja siihen ei kuulu menetelmien joustavaa valintaa. Ratkaisutapa valitaan ongelman ulkoisten piirteiden perusteella esim. sanallisissa tehtävissä menetelmän valinta avainsanan löytämisen perusteella. Käsitteellisen ymmärtämisen -tasolla opiskelija tietää miten algoritmi on seurausta matematiikan osa-alueen rakenteesta. Hän voi näin luokitella algoritmeja sen mukaan mihin aiheeseen ne liittyvät. Oppilas voi näin valita joustavasti tilanteen tai tehtävän mukaan sopivimman algoritmin. Algoritmit toimivat myös mallina muunnosten tai uusien keksimiselle. Arvioinnin ja luonnin -tasolla oppilas pystyy arvioimaan algoritmien paremmuutta tiettyyn tehtävään sekä yleistää algoritmin samantyyppisten ongelmien ratkaisemiseksi. Algoritmin yleistäminen on uuden luomista. Oppilaat voivat myös keksiä uusia algoritmeja esim. yhdistämällä aikaisemmin opittuja algoritmeja. Arviointi ja luominen ovat lähtökohtana kehittyneelle joustavuudelle, jossa pystytään perustellusti valitsemaan algoritmi, joka sopii ongelmaan parhaiten.

*Tässä esitetty joustavan matemaattisen ajattelun kuvaus on sama kuin "JoMa B: Joustavan matemaattisen ajattelun tukeminen lukiossa" -kurssilla käytetty.

4 Matemaattisesta ongelmanratkaisusta tietokonemalliin

Kun ongelma on ensin kuvattu täsmällisesti, voidaan se mallintaa matemaattisesti (ks. Kuvio 4.1 nuoli 1 ongelmasta matemaattiseen malliin). Mallinnettaessa luodaan ongelmaa kuvaava matemaattinen kaava, jonka muuttujat ja arvot edustavat ongelman tekijöitä. Operaattorit edustavat ongelman tekijöiden välisiä suhteita. Ongelman ratkaisu on kaavan toteuttavan algoritmin laskemista (ks. Kuvio 4.1 nuoli 2 matemaattisesta mallista ratkaisuun). Eri laskutavat edustavat eri algoritmeja, jolloin kaavan operaatiot tehdään eri järjestyksessä. Saman ongelman ratkaisuun voi löytyä myös erilaisia kaavoja. Nämä perustuvat siihen, että matematiikan osa-alueessa voi olla erilaisia käsitteiden yhdistelmiä, joilla voidaan esittää sama asia.



Kuvio 4.1 (sama kuin Kuvio 1.1) Matemaattista ja tietokoneen ohjelmointiin perustuvaa ongelmanratkaisua yhdistää algoritmin laskeminen (2). Lähtökohtana on ongelman kuvaus, jonka sisältämistä käsitteistä ja niiden välistä suhteista luodaan ongelmaa kuvaava kaava (1) eli ongelman matemaattinen malli. Kaavan voi ratkaista laskemalla käsin (2), mutta jos laskeminen on hankalaa tai dataa on paljon, voidaan käyttää tietokonetta apuvälineenä. Silloin luodaan tietokonemalli kaavasta ohjelmoimalla (1). Tietokoneen ohjelman suorituksen (2) tulosta voi hyödyntää (3) sekä kokonaisena että osittaisena matemaattisena ratkaisuna. Mahdollisen matemaattisen käsittelyn jälkeen pitää ratkaisu vielä tulkita ongelman kontekstissa (3), ennen kuin ongelma on ratkaistu käytännössä (4).

Edellä on esitetty kolme matemaattisen algoritmin ymmärtämisen tasoa: tiedon ja taidon, käsitteellisen ymmärtämisen sekä arvioinnin ja luomisen. Algoritmien ymmärtäminen vaikuttaa myös kykyyn ratkoa ongelmia. Tiedon ja taidon tasolla on ongelman muistutettava oppilaan aikaisemmin kohtaamaa (tieto), jotta hän voi ratkaista sen tuntemansa kaavan (tieto) ja sen ratkaisevan algoritmin avulla (taito). Käsitteellinen ymmärtäminen mahdollistaa monimutkaisemmasta ongelmasta tutun rakenteen havaitsemisen tai ongelman muokkaamisen sen esille tuomiseksi. Kyky arvioida algoritmien paremmuutta ja luoda omia algoritmeja on ongelmanratkaisun korkein taso.

Vaikka matemaattinen ratkaisuperiaate tiedetään, voi laskun monimutkaisuus tai datan määrä olla sellainen, että laskutoimitusta (algoritmia) ei pystytä suorittamaan järkevässä ajassa. Tietokoneet kehitettiin laskemaan ihmistä tehokkaammin ja ohjelmoimalla voidaan algoritmi kuvata tietokoneen ymmärtämässä muodossa. Tekoälyn kehittymisen myötä voidaan ongelmia myös ratkaista koneoppimisella esimerkiksi silloin, kun data on liian epätasaista perinteisin menetelmin määriteltäväksi tai on pystyttävä käsittelemään automaattisesti myös sellaisia tapauksia, joita ei

tiedetä etukäteen. Koneoppimisessa tietokone opetetaan malli-datalla tunnistamaan haluttuja piirteitä, jolloin varsinaisesta datasta voidaan löytää samankaltaisia asioita.

Matemaattisen mallin (kaavan) voi muuntaa tietokonemalliksi määrittelemällä sitä laskevan ohjelman (ks. edellä Kuvio 4.1 nuoli 1 matemaattisesta mallista tietokonemalliin). Vaihtoehtoisesti tietokoneelle voidaan delegoida vain laskutoimituksen osa (ks. edellä Kuvio 4.1 matemaattisesta mallista lähtevästä nuolesta 2 tietokoneeseen lähtevä nuoli 1). Algoritmia ohjelmoitaessa on otettava huomioon tietokoneen ominaisuudet (mm. laskuoperaatiot, ohjausrakenteet, muistinkäsittely sekä syöte ja tuloste) laskennan automatisoimiseksi.

5 Ongelmanratkaisu ja algoritminen ajattelu ohjelmoinnissa

Tarkasteltaessa edellä algoritmien arviointia (arvioinnin ja luomisen taso) huomattiin, että sama laskutoimitus (kaava) voidaan toteuttaa erilaisilla algoritmeilla. Algoritmi voidaan myös toteuttaa erilaisilla menetelmillä (esim. symbolinen ja visuaalinen laskenta). Ohjelmointi tarjoaa yhden menetelmän lisää ongelmanratkaisuun, joka myös automatisoi laskennan. Se poikkeaa matematiikan laskemisesta, koska laskemisen sijaan algoritmi määritellään eksplisiittisesti. Ohjelmointi muistuttaa siksi enemmän matemaattista päättelyä tai todistamista. Samantyyppisiä laskutoimituksia laskevassa ohjelmassa on lisäksi huomioitava, millä syötteiden arvovälillä ohjelman on tarkoitus toimia.

Algoritmien ohjelmoinnin ymmärtämistä voidaan myös tarkastella vastaavasti kuin matematiikan laskutoimituksia. Carsten Schulten (2008) luoma ohjelmoinnin lohkomalli (ks. Taulukko 5.1) jakaa ohjelman ymmärtämisen kolmeen luokkaan: ohjelmakoodi, ohjelman suoritus ja ohjelman tarkoitus. Kaksi ensimmäistä luokkaa liittyy ohjelman rakenteeseen ja kolmas ohjelman toimintaan. Luokat on vielä jaoteltu tarkastelutasoihin: atomi, lohko, lohkojen suhteet ja makro. Lohkomallin avulla voidaan analysoida oppilaan ohjelmakoodin ja sen merkityksen ymmärtämistä sekä havaita näiden puutteita.

Taulukko 5.1 Ohjelmoinnin ymmärtämisen lohkomalli (Schulte 2008).

	Rakenne		Toiminnallisuus
	Ohjelmakoodi	Ohjelman suoritus	Ohjelman tarkoitus
Makrotaso	Ohjelman rakenteen ymmärtäminen	Ohjelman/algoritmin suorituksen ymmärtäminen	Ohjelman tarkoituksen ymmärtäminen
Lohkojen suhteet	Lohkojen väliset suhteet (esim. funtiokutsut tai jaetut muttujat)	Lohkojen suoritusjärjes	Mikä on toisen lohkon kontribuutio tarkasteltavana olevan lohkon tarkoitukseen
Lohkot	Ohjelmointikielen käsitteistä koostuvia lauseita tai lauseiden muodostamia kokonaisuuksia	Lohkon suorituksen ymmärtäminen	Lohkon tarkoitus (osatavoite)
Atomit	Ohjelmointikielen käsitteet	Lauseen (pienin suoritettavissa oleva yksikkö) suorittaminen	Lauseen suorituksen tarkoitus

Ohjelmakoodi

Ohjelmakoodin luokassa keskitytään ohjelman tekstiin (matemaattisen) algoritmin ja tietokoneen ohjaamiseen tarvittavien toimintojen kuvaajana. Oppilaan on tunnistettava ohjelmointikielen käsitteet (atomit), jotta hän voi niistä koostaa järkeviä lauseita (lohko). Samaan tehtävään liittyvät lauseet muodostavat isomman lohkon ja usein ohjelmointikielessä on myös valmiita käsitteitä lohkojen esille

tuomiseksi (esim. ehto- ja toistolauseet tai aliohjelmat kuten proseduurit sekä funktiot). Kahden lohkon välillä voi olla suhde: esim. yhdestä lohkosta kutsutaan toisaalla määriteltyä funktion muodostavaa lohkoa. Makrotasolla teksti kuvaa ohjelman rakennetta, joka koostuu lohkoista ja yksittäisistä lauseista.

Ohjelmointikielen käsitteiden käyttöä ja ohjelman tekstin lukemista voidaan harjoitella samoin kuin matematiikkaa tiedon ja taidon tasolla. Opettaja esittää esimerkein, miten käsitteitä käytetään ja oppilaat ratkaisevat tämän jälkeen vastaavia tehtäviä. Myös valmiiden ohjelmien merkityksen tulkinta sopii ohjelmointikielen käsitteiden käytön oppimiseen. Matemaattiset algoritmit sopivat hyvin yksinkertaisiksi ohjelmointitehtäviksi, joissa haasteena on esittää intuitiiviset valinnat ja proseduurit eksplisiittisinä käyttäen ohjelmointikielen käsitteitä.

Ohjelman suoritus

Ohjelmointi on tietokoneen ohjaamista ja vaikka ohjelmointikielet sopivat hyvin matemaattisten algoritmien esittämiseen, voi taustalla oleva ohjelmointikielen suorituksen malli aiheuttaa aluksi oppilaalle yllätyksiä. Ohjelmointikielen suorituksen malli (Sorva 2018) on tietokoneen yksinkertaistettu esitys, jolla sen ominaisuuksia voidaan hyödyntää. Jokaisella ohjelmointikielellä on oma mallinsa, mutta ne voidaan luokitella mm. sen mukaan ovatko ne lähempänä teknistä toteutusta vai matemaattista ongelmaa. Imperatiiviset kielet, kuten Java, Python ja Scratch ovat lähempänä tietokonetta kun taas funktionaaliset kielet, kuten Haskell, Ocaml ja Racket ovat lähempänä matematiikkaa. Suomessa ohjelmoinnin opetuksen painopiste on imperatiivisissa kielissä, kuten Python.

Puutteet ohjelmointikielen suorituksen mallin ymmärtämisessä voivat aiheuttaa opiskelijoille väärinkäsityksiä ohjelman toiminnan tulkinnassa (Sorva 2018). Haasteeksi muodostuu, että ohjelmointikielen suorituksen mallia ei voi päätellä itse kielestä, koska se edustaa taustalla olevaa tietokoneen toimintaa. Siksi mallin esittäminen oppilaille ja ohjelman “suorittaminen” hyödyntäen mallia, voi selkeyttää ohjelman ymmärtämistä. Imperatiivisia kieliä varten on kehitetty yksinkertainen tietokoneen muistin käsittelyyn perustuva malli (Dragon & Dickson 2016). Alkuperäisessä mallissa muistin rakenne edustaa pinoa, jossa uuden muuttujan arvo laitetaan aina edellisen muuttujan yläpuolelle. Tässä on pinorakenne jätetty pois, koska sitä ei tarvita yläkoulun matematiikan ohjelmoinnin kontekstissa. Muokattu malli kuvaa muistia kaksisarakkeisena taulukkona, jossa vasemmalle on muuttujien nimet ja oikealla niiden arvot (ks. Kuvio 5.1). Tarkoituksena on käydä ohjelmaa lävitse lause kerrallaan merkitsemällä muistiin uudet muuttujat ja niiden arvot sekä korvata yliviivaamalla muuttujien vanhat arvot, kun niihin asetetaan uusia. Taulukkoa täydennetään ylhäältä alas. Lisäksi tulosteeseen merkitään tulostuslauseiden sisällöt allekkain.

Muuttujan arvon asettaminen

```
s = "hello"
x = 4
y = 6
x = 8
total = x + y + x
print(total)
```

muuttujan nimi	arvo
s	“hello”
x	~~4~~ 8
y	6
total	22

Tuloste

```
22
```

Kuvio 5.1 Ohjelmointikielen suorituksen malli, joka kuvaa ohjelman muistin käsittelyä. Muisti kuvataan taulukkona, jossa vasemmalla on muuttujien nimet ja oikealla muistissa niitä vastaavat arvot. Muuttuja merkitään taulukkoon siinä järjestyksessä, kun ne esiintyvät ohjelmassa. Jos aikaisemman muuttujan arvoa muutetaan, niin silloin yliviivataan edellinen arvo ja laitetaan sen vierelle uusi (muuttujassa voi olla vain yksi arvo kerrallaan!).

Ohjelmointikielen suorituksen malli tuo esille sen, että kieli edustaa tietokoneen toimintaa, vaikka ohjelmointikielellä voidaan määritellä laskutoimituksia. Mallin toimintaa voidaan harjoitella suorittamalla käsin ohjelma ja kirjaamalla sen aiheuttamat muutokset. Muutoksia ovat muuttujan arvon lisääminen muistiin, arvon muuttaminen (oikeammin vaihtaminen) ja tuloste. Oppilas oppii näin ymmärtämään tietokoneen toimintaa korkealla tasolla ja miten ohjelmointikielen käsitteet ohjaavat sitä*. Verrattuna algoritmisen ajattelun käsitteellisen ymmärtämisen tasoon matematiikassa huomaamme, että ohjelmointikielen malli on aina sama, riippumatta siitä, mitä algoritmi edustaa. Siksi ongelmanratkaisun sijaan, ohjelmoinnissa on kyse ratkaisun toteuttamisesta.

**Tässä on esitelty atomisten arvojen asettaminen muuttujiin (myös lausekkeiden laskutoimitusten tulokset ovat tällaisia), arvojen päivittäminen ja tuloste. Mallia voi myös laajentaa funktioihin ja tietorakenteisiin kuten listat sekä merkkijonot. Näissä muistissa on viittaus toiseen (ali)ohjelmaan tai tietojen joukkoon. Myös suoritettava ohjelma sijaitsee muistissa, onko kyseessä ohjelmakoodi vai tieto riippuu siitä, mitä tarkoitusta varten muistin osa on varattu. Tämä tuo esille muistin keskeisyyden ohjelman ja tietokoneen toiminnassa.*

Ohjelman tarkoitus

Ohjelmoinnin ymmärtämisen lohkomallin kolmannessa luokassa kohteena on ohjelman tarkoitus. Tarkoitus muodostuu alkaen yksittäisistä lauseista (atomi), millaisia lohkoja nämä muodostavat (lohko), missä järjestyksessä eri lohkot suoritetaan (lohkojen suhteet) ja lopuksi millainen kokonaisuus syntyy lohkojen suorittamisesta (makro). Lauseiden ja niistä muodostettujen lohkojen merkitys tulee niiden käyttämistä tietokoneen ominaisuuksista. Tietojenkäsittelyn mekanismien (Computing mechanics; Denning 2003) avulla voidaan luokitella ominaisuudet: laskentaan, viestintään, koordinaatioon, automaatioon ja muistamiseen liittyviksi (ks. Kuvio 5.2).

Kuvio 5.2 Tietojenkäsittelyn mekanismit tarjoavat viisi tietojenkäsittelytieteen näkökulmaa. Näkökulmia voi käyttää kuvaamaan tietokoneen ja siihen liittyvien teknologioiden hyödyntämisen mahdollisuuksia.

Tietojenkäsittelyn viittä mekanismia voidaan käyttää kuvaamaan tietojenkäsittelyyn liittyviä teknisiä ominaisuuksia. Yleensä jokin ominaisuus sisältää useamman mekanismin piirteet siksi mekanismit muistuttavat enemmän näkökulmia, joita voidaan yhdistää tiettyä teknistä ratkaisua tarkasteltaessa. Olemme tähän mennessä ohjelmoinnissa jo viitanneet automaatioon, muistamiseen, laskentaan ja viestintään (ks. edellä Kuvio 5.2). Ohjelma suoritetaan automaattisesti. Muuttujien hyödyntäminen on muistin käsittelyä ja jos tallennamme ohjelman tietokoneelle, on kyseessä pitkäaikaismuisti kuten kovalevy. Laskutoimituksessa (ks. edellä Kuvio 5.2) laskettiin kolmen muuttujan arvot yhteen ja laskutoimituksen tulos sijoitettiin uuteen muuttujaan. Print-funktion käyttö on viestintää koneesta ulospäin: tällöin näemme tulosteen ruudulla. Koordinaatio koskee koordinaatiota ohjelmien, tietokoneiden, ihmisten tai näiden yhdistelmien välillä.

Ohjelman tarkoituksen määrittely on tietokoneen eri ominaisuuksien yhdistämistä oikeanlaisella tavalla ongelman ratkaisemiseksi. Ratkaisu ei tällöin ole enää matemaattinen vaan tekninen ja ongelman mallintaminen muuttuu tietokoneella toteutuksen suunnitteluksi. Matemaattisia kaavoja toteuttavat algoritmit voidaan muuntaa suhteellisen suoraviivaisesti tietokoneelle. Ongelmat, jotka vaativat joukon erilaisia laskutoimituksia vaativat jo enemmän suunnittelua. Simulaatioissa on usein huomioitava monen eri osatekijän yhteisvaikutus. Tekoälyyn (mm. koneoppiminen) tarvitaan siihen liittyvien tietokonetta ohjaavien algoritmien tuntemusta. Aina ei ohjelmia tarvitse kirjoittaa uudestaan vaan voidaan kirjoittaa ohjelmia, jotka toimivat työkaluina (yleiskäyttöiset ohjemalohkot kuten funktiot) samantyyppisten ongelmien ratkaisemiseen (ks. edellä Taulukko 5.1).

Ohjelman tarkoituksen luokassa ohjelmointi on luovaa suunnittelua. Matemaattisen kaavan toteuttamisen lisäksi siihen sisältyy tietokoneen ominaisuuksien hyödyntämistä, jotta ohjelmaa olisi kätevä käyttää. Kun ohjelman ominaisuudet kasvavat, voidaan ohjelmaa ajatella algoritmina, joka koordinoi ohjelman lohkojen toimintaa ja siirtymistä niiden välillä. Ohjelmistot ovat monesta ohjelmasta koostuvia kokonaisuuksia ja niiden hallintaan tarvitaan teknisempiä ratkaisuja. Yläkoulun ohjelmoinnissa ei kuitenkaan mennä näin pitkälle. Oppilaan tulisi kuitenkin pystyä suunnittelemaan tietokonetta ohjaava ohjelma, joka mahdollistaa erillisistä lohkoista koostuvan kokonaisuuden suorittamisen. Näin voidaan laskutoimituksen lisäksi hyödyntää tietokoneen eri ominaisuuksia ohjelmointikielen mukana tulevien aliohjelmien avulla.

6 Joustava matematiikka ja ohjelmointi

Matemaattisesta kontekstista riippuen joustavuudella on erilaisia ilmenemismuotoja*

- ratkaisustrategian valinnassa ja
- esitysmuotojen käytössä.

Ratkaisustrategian tai algoritmin valintaan vaikuttaa myös millaista välinettä sen toteuttamiseen käytetään. Päässälaskussa pyritään helpommin muistettavaan algoritmiin, mutta jos käytetään esim. paperia ja kynää voidaan hyödyntää monimutkaisempia proseduureja. Paperi toimii silloin ulkoisena muistina, jota voidaan hyödyntää laskun edetessä. Erilaisten laskutoimitusten suorittamisen apuvälineiden joukossa tietokone on erityinen, koska algoritmin määrittelyn jälkeen se automatisoi laskemisen. Automaation avulla voidaan laskea laskutoimituksia, jotka ovat liian monimutkaisia ihmiselle tai käsitellä data-määriä, jotka eivät olisi käsin mahdollisia. Algoritmi määritellään tietokoneella käyttäen ohjelmointikieltä. Ohjelmointikieliä on monenlaisia, joidenkin käsitteet ovat lähempänä matematiikkaa, kun taas toiset kuvaavat tietokonelaitteiston ominaisuuksia.

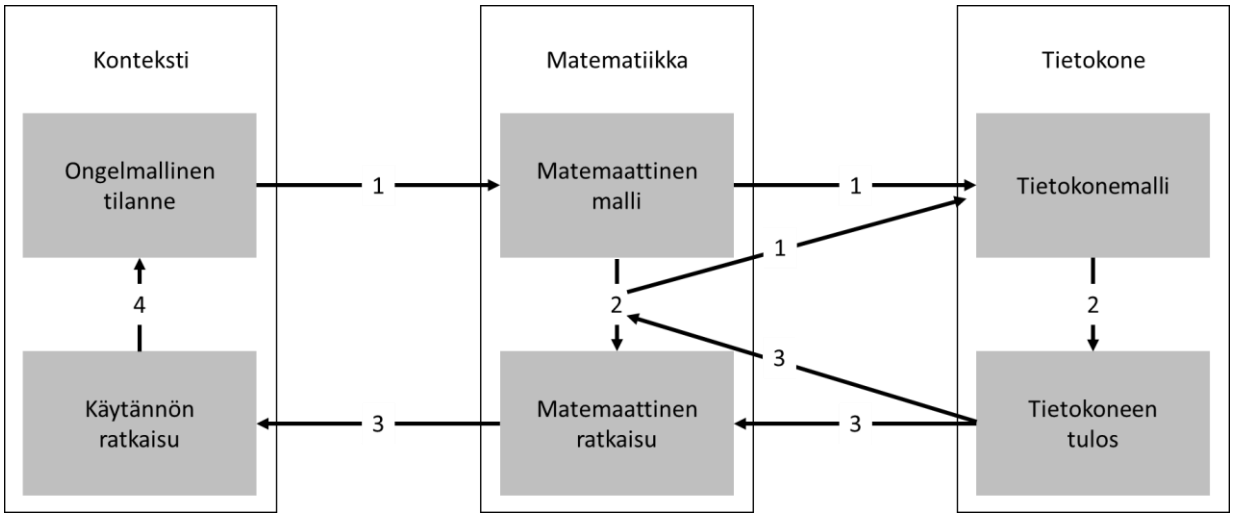
Suomen kouluissa käytetyt ohjelmointikielet ovat lähempänä tietokonelaitteistoa. Niitä kutsutaan imperatiivisiksi kieliksi, koska ne koostuvat käskyistä, jolla tietokonetta ohjataan. Yksi esimerkki tällaisesta ohjelmointikielestä on Python kieli (https://www.python.org), joka on myös suosittu kieli opetuskäyttöön. Imperatiiviset kielet keskittyvät muistin käsittelyyn ja tämän käsittelyn ohjaamiseen peräkkäis-, ehto- ja toistorakenteilla. Näiden kielten suorituksen malli on yksinkertaistus tietokoneen toiminnasta. On olemassa toinenkin luokka kieliä, joissa malli on otettu matematiikan funktioista ja joita siksi kutsutaan funktionaalisiksi kieliksi. Funktionaalisissa kielissä ohjelma koostuu sisäkkäisistä funktiosta ja ohjelman suoritus on sieventämistä. Yksi esimerkki funktionaalisesta kielestä on Racket (racket-lang.org), jota on myös käyttetty yläkoulun ohjelmoinnin opetukseen. Kyky valita sopiva ohjelmointikieli ongelman mukaan, on esimerkki joustavuudessa vaikka siihen voi vaikuttaa myös muut kuin matemaattiset tekijät.

Joustavaa matemaattista ajattelua tarvitaan ohjelmoinnissa parhaimman algoritmin valitsemiseen. Algoritmien paremmuutta mitataan yleensä laskemalla, kuinka monta kertaa ohjelman lauseita suoritetaan tai kuinka paljon muistia on varattava. Lauseiden suoritusmäärä on näistä yleisempi ja se edustaa algoritmin tehokkuutta. Tehokas algoritmi on luotu niin, että suoritetaan vähemmän lauseita, kuin ongelman koon perusteella voisi ajatella. Esimerkkinä tällaisesta voidaan pitää puolitushakualgoritmia, jossa lajitellusta yksiulotteisesta taulukosta valitaan aina keskimmäinen alkio ja riippuen onko se isompi vai pienempi kuin haettava luku, valitaan vasen tai oikea puoli. Sama proseduuri toistetaan, kunnes jäljellä on vain yksi alkio, joka on vastaus. Puolitushakualgoritmi käy lävitse huomattavasti pienemmän määrän alkioita, kuin koko taulukon läpikäynnissä. Joustavalla matemaattisella ajattelulla on merkitystä, koska huonosti valittu algoritmi voi isoissa ongelmissa tehdä ratkaisun tietokoneellakin mahdottomaksi.

*Tässä esitetty joustavan matemaattisen ajattelun kuvaus on sama kuin "JoMa B: Joustavan matemaattisen ajattelun tukeminen lukiossa" -kurssilla käytetty.

7 Algoritminen ajattelu ja ongelmanratkaisu

Edellä on esitelty kolme ohjelmoinnin ymmärtämisen luokkaa tulkintana algoritmisesta ongelmanratkaisusta tietokoneella. Ohjelmakoodin luokassa keskitytään ohjelman tekstiin algoritmin kuvaajana. Ohjelman suoritus nostaa esille sen, että tietokoneella algoritmin esitys viittaa koneen toiminnan ohjaamiseen. Ohjelman tarkoitus liittää ongelman tietokoneen ominaisuuksien yhdistelmän tarjoamaan ratkaisuun. Tietokoneen tuloste on ratkaisu esitettyyn ongelmaan (ks. Kuvio 7.1). Se voi vaatia vielä matemaattista tulkintaa tai käsittelyä ennen kuin se voidaan asettaa alkuperäisen ongelman kontekstiin sen ratkaisemiseksi.



Kuvio 7.1 (sama kuin Kuvio 1.1 ja 4.1) Ohjelmoimalla toteutettu algoritmi on ongelman malli (1), jonka laskemisen (2) tietokone automatisoi. Tietokoneen tulosta voi hyödyntää (3) sekä kokonaisena että osittaisena matemaattisena ratkaisuna. Mahdollisen matemaattisen käsittelyn jälkeen pitää ratkaisu vielä tulkita ongelman kontekstissa (3), ennen kuin ongelma on ratkaistu käytännössä (4).

Vaikka ohjelmakoodi antaisi ymmärtää, että matemaattinen ja tietotekninen ongelmanratkaisu olisi samanlaista, tarkempi tarkastelu (matematiikan algoritmisen ajattelun tasojen ja ohjelmoinnin ymmärtämisen lohkomallin luokkien vertailu) tuo esille niiden erilaisuuden. Ohjelmakoodin tarkoituksena on ohjata tietokoneen toimintaa ja hyödyntää sen ominaisuuksia. Korkean tason ohjelmointikielten tarjoamat matemaattiset symbolit ovat valmiita teknisiä toteutuksia. Kun toimitaan näiden puitteissa ohjelmointikielen lausekkeissa, voidaan luottaa siihen, että ne käyttäytyvät matemaattisesti oikein. Matemaattisten lausekkeiden ulkopuolella arvojen käsittely perustuu tekniseen malliin, joka on huomioitava, kun yhdistetään laskutoimituksia ohjelmointikielen muihin käsitteisiin.

Tietokone luotiin matematiikan työvälineeksi. Matematiikasta on hyötyä tietokoneen ohjelmoinnissa, vaikka kohteena oleva ongelma ei itsessään olisi matemaattinen, koska de tarjoaa valmiita ratkaisuja monenlaisten säännönmukaisuuksien käsittelyyn. Yhtäläisyyksistä huolimatta, algoritmit matematiikassa ja ohjelmoinnissa ovat erilaisia. Ohjelmointia opetettaessa sekä yhtäläisyydet että erot on tehtävä selkeäksi, jotta vältytään väärinkäsityksiltä.

Lähteet

Denning, P. J. (2003). Great principles of computing. Communications of the ACM, 46(11), 15-20.

Dragon, T., & Dickson, P. E. (2016). Memory Diagrams: A Consistant Approach Across Concepts and Languages. In Proceedings of the 47th ACM Technical Symposium on Computing Science Education (pp. 546-551).

Hoyles, C. (2018). Transforming the mathematical practices of learners and teachers through digital technology. Research in Mathematics Education, 20(3), 209-228.

Fan, L., & Bokhove, C. (2014). Rethinking the role of algorithms in school mathematics: A conceptual model with focus on cognitive development. ZDM, 46(3), 481-492.

Joutsenlahti, J., & Tossavainen, T. (2018). Matemaattisen ajattelun kielentäminen ja siihen ohjaaminen koulussa. In Matematiikan opetus ja oppiminen (pp. 410-431).

Kallia, M., van Borkulo, S. P., Drijvers, P., Barendsen, E., & Tolboom, J. (2021). Characterising computational thinking in mathematics education: a literature-informed Delphi study. Research in Mathematics Education, 1-29.

Morgan, J. (2019). A Compendium of Mathematical Methods. John Catt Educational.

OPH. (2014) Perusopetuksen opetussuunnitelman perusteet. Opetushallitus.

Rich, K. M., & Yadav, A. (2020). Applying levels of abstraction to mathematics word problems. TechTrends, 1-9.

Schulte, C. (2008). Block Model: an educational model of program comprehension as a tool for a scholarly approach to teaching. In Proceedings of the Fourth international Workshop on Computing Education Research (pp. 149-160).

Sorva, J. (2018). Misconceptions and the Beginner Programmer. Computer Science Education: Perspectives on Teaching and Learning in School, 171.