OHJELMOINNILLISEN AJATTELUN TYÖKALUT ALAKOULUSSA

Turun yliopiston Oppimisanalytiikan keskuksen järjestämän täydennyskoulutuskurssin sisällön kuvaus

OHJELMOINNILLISEN AJATTELUN TYÖKALUT ALAKOULUSSA

TURUN YLIOPISTON OPPIMISANALYTIIKAN KESKUKSEN 2021 JÄRJESTÄMÄN TÄYDENNYSKOULUTUSKURSSIN SISÄLLÖN KUVAUS

Peter Larsson

Heidi Kaarto

Marika Parviainen

CC BY-SA 4.0

Oppimisanalytiikan keskus

oppimisanalytiikka.fi

Tämä työ on julkaistu Creative Commons lisenssillä CC BY-SA 4.0
(https://creativecommons.org/licenses/by-sa/4.0/deed.fi), jollei sisällön yhteydessä muuta mainita.

Tekijänä on Turun yliopiston Oppimisanalytiikan keskus ja lisätietoa löytyy osoitteesta oppimisanalytiikka.fi, jollei sisällön yhteydessä muuta mainita.

SISÄLLYS

1 Johdanto 1
1.1 Kurssin kuvaus 1
1.2 Kurssin tekijät 2
2 Verkkosisältö 1: Matematiikka, ohjelmointi ja ohjelmoinnillinen ajattelu 4
2.1 Ohjelmointi ja matematiikka 5
2.2 Matemaattinen ongelmanratkaisu 9
2.3 Algoritminen ongelmanratkaisu 12
2.4 Ohjelmoinnillinen ajattelu ongelmanratkaisussa 16
3 Työpaja 1: Ohjelmoinnillisen ajattelun harjoituksia alkuopetukseen 19
4 Verkkosisältö 2: Tietokoneen mahdollisuuksista ohjelmointiin 21
4.1 Tietokone ja ohjelmointi 21
4.2 Ohjelmoinnilliset/algoritmiset pelit 24
4.3 Graafiset ohjelmointiympäristöt 29
4.4 Tutustutaan ohjelmoinnin konsepteihin 31
5 Työpaja 2: Tutustutaan graafiseen ohjelmointiin 37
6 Verkkosisältö 3: Scratch-ohjelmointi ja sen soveltaminen matematiikassa 40
6.1 Tietokoneen hyöty matematiikalle 40
6.2 Scratch-ympäristön esittely 43
6.3 Ohjelmoinnin konseptit Scratchissä 46
6.4 Ohjelmoinnin konseptit jatkuvat 55
6.5 Ohjelmoinnin soveltaminen matematiikassa 59
7 Työpaja 3: Scratch-ohjelmointi 64
8 Verkkosisältö 4: Ohjelmoinnin opetuksen menetelmiä 66
8.1 Ensin unplugged, sitten ohjelmointi 66
8.2 TIPP&SEE menetelmä ohjelmoinnin oppimisen tukemiseen 70
8.3 5E-menetelmä matematiikan ja ohjelmoinnin yhdistämiseen 73
8.4 Erilaisia ohjelmoinnin tehtäviä 76
9 Työpaja 4: Ohjelmoinnin opetus 84
10 Yhteenveto 87

1 JOHDANTO

Tämä on kuvaus Turun yliopiston Oppimisanalytiikan keskuksen toteuttaman ja Opetushallituksen rahoittaman Ohjelmoinnillisen ajattelun työkalut perusopetuksessa -projektin ensimmäisestä täydennyskoulutuksesta. Projektissa suunniteltiin ja toteutettiin kaksi täydennyskoulutuskurssia: toinen alakoulun ja toinen yläkoulun opettajille. Tämä on kuvaus alakoulun kurssista, joka tarjoaa tarvittavat pohjatiedot ohjelmoinnin opettamiseen osana alakoulun matematiikkaa.

Ohjelmoinnin opetus voi tuntua haastavalta ja vaikka erilaista materiaalia on tarjolla, on vaikeaa tehdä pedagogisesti perusteltuja valintoja. Tämän kurssin tavoitteena oli antaa opettajalle perustiedot ohjelmoinnista ja valmiudet matematiikkaa soveltavien ohjelmoinnin materiaalien valitsemiseen ja luomiseen. Kurssilla huomioitiin erityisesti opetussuunnitelman määrittelemät vaiheittaisten toimintaohjeiden käytön oppiminen alkuopetuksessa ja näiden toteuttaminen graafisessa ohjelmointiympäristössä luokilla 3–6 tavoitteet. Ohjelmoinnillinen ajattelu antaa opettajalle työkalut ohjelmoinnin oppimisen ymmärtämiseen ja alakoulun tavoitteiden saavuttamiseen.

1.1 KURSSIN KUVAUS

Kurssin laajuus oli 2 opintopistettä, ja se koostui aloitusluennosta, neljästä verkkosisällöstä ja neljästä työpajasta. Tässä koosteessa on verkkoluentojen sisältö ja kuvaukset työpajoista.

1. Aloitusluento: Miksi ohjelmointia alakoulussa?
2. Verkkosisältö 1: Matematiikka, ohjelmointi ja ohjelmoinnillinen ajattelu
3. Työpaja 1: Ohjelmoinnillisen ajattelun harjoituksia alkuopetukseen
4. Verkkosisältö 2: Tietokoneen mahdollisuuksista ohjelmointiin
5. Työpaja 2: Tutustutaan graafiseen ohjelmointiin
6. Verkkosisältö 3: Scratch-ohjelmointi ja sen soveltaminen matematiikassa
7. Työpaja 3: Scratch-ohjelmointi
8. Verkkosisältö 4: Ohjelmoinnin opetuksen menetelmiä
9. Työpaja 4: Ohjelmoinnin opetus

Kurssin eri osiot aukesivat portaittain, noin kahden viikon välein. Osallistujille suositeltiin uuteen osioon tutustumista heti osion auettua, jotta pysyy mukana keskustelussa ja ehtii tutustua teoriapohjaan ennen työpajaa. Verkkosisältöjen tehtävien toivottiin olevan tehtyinä ennen seuraavaa verkkoluentoa.

Verkkosisällöt koostuivat teoriasta ja tehtävistä, joita tarjottiin digitaalisen ViLLE-oppimisympäristön kautta. Työpajat järjestettiin etätapaamisina Zoom-palvelussa ja osallistujilta edellytettiin paikalla oloa. Aloitusluento järjestettiin myös etänä Zoomissa, ja se oli saatavilla tallenteena ViLLEssä jälkeenpäin. Kotitehtävät ja projektit tehtiin aidossa ohjelmointiympäristössä, mutta tuotokset palautettiin ViLLEen.

Kurssin lopulla tuotiin teoria käytäntöön ja luotiin harjoitustyönä oma opetustuokio tai oppituntisuunnitelma ohjelmoinnin opetukseen. Suunnitelmat palautettiin ja niihin sai asiantuntijakommentteja.

Kurssin suorittamiseksi hyväksytysti osallistujilta vaadittiin 75 % prosenttia ViLLEssä saaduista pisteistä ja loppuprojektin palauttaminen. ViLLEn pisteet muodostuvat verkkosisältöjen tehtävistä, työpajojen kotitehtävistä, loppuprojektista sekä erilaisista kyselyistä saaduista pisteistä.

1.2 KURSSIN TEKIJÄT

Kuva 1: Peter Larsson, Heidi Kaarto ja Marika Parviainen

Peter Larsson toimii tietotekniikan laitoksella opettajana ja tutkijana. Opetus koskee digitaalisia teknologioita ja yhteiskunnan digitalisaatiota. Tutkimus puolestaan keskittyy ohjelmoinnin ja ohjelmoinnillisen ajattelun opettamiseen peruskoulussa. Keskeisenä kysymyksenä on miten yhdistää ohjelmointi matematiikan opetukseen, niin että kumpikin aihe hyötyy. Ohjelmoinnin opetus peruskoulussa on vielä uutta, joten on tärkeää kehittää opetuksen sisältöjä ja menetelmiä. Peterin vapaa-aikaan kuuluu luonnossa liikkuminen, uinti, lauta- ja tietokonepelit sekä lukeminen.

Heidi Kaarto on suunnitellut ja toteuttanut useita peruskoulun ja lukion ohjelmointikursseja ja opettanut ohjelmointia lukiossa ja yliopistossa. Hän on aina halunnut olla opettaja ja lemmikinomistaja.

Marika Parviainen on IT-laitoksen kasvatti ja pitkän linjan villeläinen, jolla on kokemusta ohjelmointitehtävien laatimisesta, koulutusten järjestämisestä sekä käyttäjätuesta.

Kurssilla oli kaksi vierailijaluentoa, joita piti Valentina Dagiene ja Anu Tuominen.

Valentina Dagiene on professori ja johtava tutkija Vilnan yliopiston datatieteen ja digitaalisten teknologioiden laitoksella. Hän on kirjoittanut yli 200 tieteellistä artikkelia ja 60 kirjaa liittyen tietojenkäsittelytieteen opetukseen. Dagiene on Informatics in Education ja Olympiads in Informatics julkaisujen päätoimittaja. Vuonna 2004 hän perusti kansainvälisen BEBRAS - tietojenkäsittelytieteen ja ohjelmoinnillisen ajattelun kilpailun, johon osallistuu oppilaita yli 60 maasta.

Anu Tuominen on työskennellyt opettajankoulutuksessa vuodesta 2006 alkaen opettaen matematiikkaa luokanopettaja-, aineenopettaja- ja erityisopettaja opiskelijoille. Tuominen kirjoittaa työnsä ohella väitöskirjaa murtolukujen oppimisesta alakoulun 3. luokalla.

Kuva 2: Rahoittajana toimi Opetushallitus, oppimisympäristönä ViLLE, toteuttajana Turun yliopiston Oppimisanalytiikan keskus ja yhteistyökumppanina Vilnan yliopisto (Liettua).

2 VERKKOSISÄLTÖ 1: MATEMATIIKKA, OHJELMOINTI JA OHJELMOINNILLINEN AJATTELU

Tässä osiossa pohditaan, miksi on perusteltua opettaa ohjelmointia matematiikan yhteydessä. Ohjelmoinnissa ja matematiikassa on paljon yhteistä, niin perusteiden kuin sovellustenkin tasolla - molemmissa tarkastellaan asioita täsmällisten määrittelyjen ja tarkkojen ohjeiden avulla. Kummassakin lasketaan ja laskutoimitusten suoritustapaa määrittelee algoritmi (ks. Kuva 2). Kuitenkin matematiikassa opimme algoritmit tapoina suorittaa laskutoimituksia, kun taas ohjelmoinnissa algoritmit ovat enemmän esillä, koska vain niiden kautta voidaan varmistua, että tietokone tekee sen mitä on suunniteltu.

Sanat algebra ja algoritmi tulevat molemmat muinaisen matemaatikon Al-Khwarizmin nimestä. Hänen matematiikkaa käsitteleviä teoksiaan käännettiin keskiajalla latinaksi ja kirjoittajan nimen latinankielinen kirjoitusasu algoritmi vakiintui merkitsemään matemaattista toimintaohjetta.

Kuva 2: Mistä tulee sana algoritmi. (Scratch cat picture, (CC BY-SA 2.0). Scratch is developed by the Lifelong Kindergarten Group at the MIT Media Lab. See http://scratch.mit.edu.)

Oppilaita varten laadituissa ohjeissa ei mainita termiä algoritmi, kysehän on toimintaohjeista. Opettajan on silti syytä tietää, miten algoritmi sijoittuu ohjelmoinnin ja matematiikan kentälle.

Algoritminen ajattelu on tärkeä osa ohjelmointiin liittyvää ajattelua. Tulevaisuuden työelämässä kaikki eivät ole ohjelmoijia, mutta ohjelmoinnillinen ajattelu auttaa digitaalisuuden ymmärtämisessä ja siihen perustuvien työvälineiden käytössä. Työelämän monitahoisia ongelmia voi ymmärtää paremmin, kun pystyy asettelemaan ajatuksensa täsmällisesti ja järjestelmällisesti ja viestimään muille yksiselitteisesti. Oppimalla ohjelmointia voidaan näitä taitoja harjoitella jo alakoulussa.

Tässä osiossa opitaan lisää ohjelmoinnillisen ajattelun ja algoritmien roolista ongelmanratkaisussa yleensä ja matematiikan opetuksessa esiintyvissä ongelmanratkaisutehtävissä erityisesti. Ohjelmoinnillisen ajattelun osa-alueita käsitellään myös tarkemmin.

Vaikka alakoulun ohjelmoinnissa liikutaan hyvin käytännönläheisellä tasolla, teoriapohja auttaa ymmärtämään käsitteitä syvällisemmin ja tekemään perusteltuja pedagogisia ratkaisuja. Myöhemmin käsitellään ohjelmoinnillisen ajattelun opetukseen liittyviä toiminnallisia harjoituksia opetukseen sopivien esimerkkien avulla.

2.1 OHJELMOINTI JA MATEMATIIKKA

Matematiikan ja ohjelmoinnin yhteys on laskemisessa. Päässälaskun, laskemisen paperilla, laskemisen laskimella (tai matkapuhelimen laskinsovelluksella) rinnalle on tullut uusi menetelmä: laskeminen ohjelmoimalla. Jokaisella menetelmällä on omat etunsa käytön ja oppimisen kannalta. Ohjelmointi on myös muuta kuin laskemista, sillä voidaan määritellä miten ohjelmat ja digitaaliset laitteet toimivat. Tietokoneen toiminta perustuu kuitenkin laskemiseen, siksi ratkaistava ongelma, mallinnettava asia tai toteutettava toiminnallisuus on esitettävä matemaattisen täsmällisesti. Ohjelmoitaessa laskeminen toteutetaan algoritmina, jota kuvataan ohjelmointikielen käsitteillä.

Vaikka on keksitty teknisiä apuvälineitä, on yhä päässälaskutaito merkki siitä, että osaa matematiikkaa. Jos ei itse osaa laskea, pystyisikö ymmärtämään mistä on kysymys? Päässälaskutaidosta on hyötyä tilanteissa, joissa apuvälinettä ei ole mukana tai sen käyttäminen olisi kömpelöä. Esimerkkejä löytyy kaupankäynnistä, rakentamisesta, erilaisista käsityötaidoista ja monesta harrastuksesta. Kuitenkin kun monimutkaisuus tai käsiteltävien asioiden määrä kasvavat, alkavat päässälaskutaidon rajat tulla vastaan. Silloin laskemisen ulkoistaminen paperille auttaa laskutoimituksen jäsentämisessä ja toimii ulkoisena muistina. Paperilla laskemisen tukemiseksi on kehitetty myös menetelmiä, joita kutsutaan algoritmeiksi.

Algoritmit luotiin helpottamaan monimutkaisten laskutoimitusten suorittamista jakamalla ne yksinkertaisemmiksi laskutoimituksiksi. Esimerkkeinä tästä ovat aritmetiikan allekkainlaskut, erilaiset kaavat ohjeineen kuten Pythagoraan lause tai toisen asteen yhtälön ratkaisukaava sekä harpin ja viivoittimen käyttäminen geometriassa. Algoritmin suorittaminen vaatii usein monta askelta, joten siksi niitä tehtiin usein paperin ja kynän avulla. Paperilla laskemisessa tarvitaan kuitenkin yhä päässälaskutaitoa. Vaihtoehtoisesti voidaan käyttää laskinta, koska kyseessä on vain laskemisen mekaaninen osa. Nykyaikaisilla laskimilla voidaan toteuttaa monimutkaisiakin kaavoja, jolloin paperia ei välttämättä tarvita. Silloin laskutoimituksen askeleiden, sen algoritmin, miettiminen auttaa laskimen käytössä.

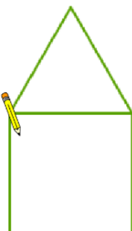
Ohjelmoinnissa algoritmi on määriteltävä täsmällisesti, koska tietokone pystyy suorittamaan vain sen tuntemat komennot. Jokainen algoritmin suorittamiseen tarvittava askel on määriteltävä. Joskus se mikä ihmiselle on vain yksi askel voi koneella vaatia useamman askeleen. Komentoja on kolmenlaisia: algoritmin etenemistä ohjaavia, laskutoimituksen suorittavia ja tietokoneen ominaisuutta hyödyntäviä. Algoritmin toteuttava ohjelma koostuu lauseista eli komennoista tai käskyistä sekä niiden tarvitsemista tiedoista. Tietokone suorittaa ohjelman lause kerrallaan esitysjärjestyksessä. Ehto- ja toistolauseet voivat kuitenkin muuttaa suoritusjärjestystä. Ehtolause määrittelee vaihtoehtoisen lauseiden ketjun, joka valitaan, jos asetettu ehto täyttyy. Ketju voi jatkua ohjelman loppuun asti tai sitten palataan pääketjuun viimeisen lauseen jälkeen. Jos samoja lauseita tarvitaan useamman kerran peräkkäin, voidaan käyttää toistolausetta, joka toistaa joukon lauseita annetun ehdon ollessa voimassa.

Ohjelman varsinainen tulos syntyy laskutoimituksia tekevillä ja tietokoneen ominaisuuksia hyödyntävillä lauseilla. Laskutoimituksia tekevät lauseet tarvitsevat syötteenä arvot (samoin kuin matematiikan funktiot), joiden perusteella laskeminen suoritetaan. Arvot annetaan yleensä asettamalla muuttujia, jotka

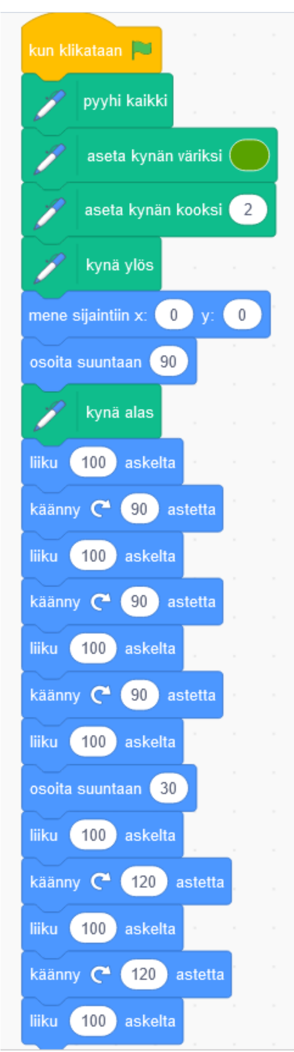
edustavat tietokoneen muistipaikkoja. Lause palauttaa laskutoimituksen tuloksen. Muuttujien arvojen asettaminen on yksi keskeisistä tietokoneen toimintaa ohjaavista lauseista. Laskutoimituksen tulos katoaa, jollei sitä tällä tavalla tallenneta. Muita laskemisessa tarpeellisia lauseita ovat syötteen lukeminen näppäimistöltä ja tuloksen tulostaminen näytölle. Vaikka tavoitteena on pelkkä laskeminen, vaatii tietokoneen hyödyntäminen myös sen muiden ominaisuuksien ohjaamista. On myös ohjelmia, jotka eivät sisällä laskutoimituksia, mutta niissäkin hyödynnetään matematiikkaa ominaisuuksien käyttöä ohjaavan algoritmin muodossa. Näin ollen ohjelma on aina yhdistelmä matematiikkaa ja tietokoneen ohjaamista (ks. Kuva 3).

Kuva 3: Scratch-ohjelma, joka piirtää viisikulmion.

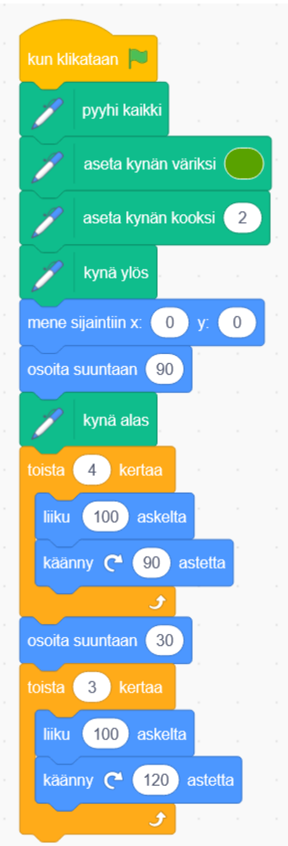
Ohjelma on aina algoritmi, mutta sana algoritmi varataan yleensä nimetyille ratkaisuille, joiden voidaan ajatella olevan laajemminkin hyödyllisiä. Erilaiset ratkaisut tietojen lajitteluun ja hakuun ovat tyypillisiä esimerkkejä usein käytetyistä algoritmeista. Samaan lopputulokseen voidaan päätyä erilaisilla algoritmeilla (ks. Kuvat 4–7). Algoritmin toteutuksessa voidaan pyrkiä yksinkertaisuuteen, nopeuteen tai mahdollisimman vähäiseen tietokoneen resurssien käyttöön. Tietokoneen ohjaamiseen tiettyä käyttötarkoitusta varten suunniteltua algoritmia kutsutaan ohjelmaksi. Ohjelman algoritmi voi sisältää useita muita algoritmeja. Nämä hoitavat tietyn määrätyn tehtävän kokonaisuudessa.



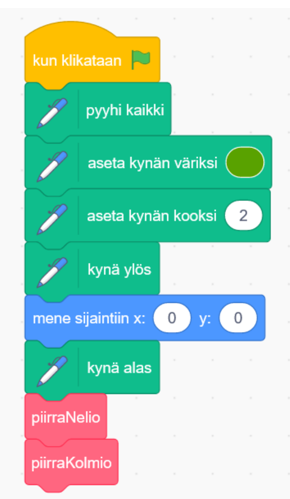
Kuva 4: Talo voidaan piirtää yhdistämällä kaksi geometristä kuviota, jossa tasasivuinen kolmio toimii kattona ja neliö talon runkona.



Kuva 5: Ensimmäisessä tavassa piirtää kolmiosta ja neliöstä koostuva talo (ks. Kuva 4) jokainen viiva on oma lauseensa ohjelmassa.



Kuva 6: Toisessa tavassa piirtää kolmiosta ja neliöstä koostuva talo (ks. Kuva 4) hyödynnetään toistoa eli katto (kolmio) ja runko (neliö) määritellään toistamalla niiden viivojen piirtämistä ja kulmien astetta määritteleviä komentoja.



Kuva 7: Kolmannessa tavassa piirtää kolmiosta ja neliöstä koostuva talo (ks. Kuva 4) hyödynnetään aliohjelmia eli kolmion ja neliön piirtäminen on kirjoitettu aliohjelmiin, joita kutsutaan pääohjelmasta (aliohjelmien määrittelemät kuviot on voitu toteuttaa edellä määritellyin tavoin).

Kun ratkaistava ongelma on monimutkainen, kasvaa ohjelman koko nopeasti isoksi. Monimutkaisuutta hallitsemaan voidaan ohjelma jakaa aliohjelmiin, joita kutsutaan funktioiksi, proseduureiksi tai metodeiksi, riippuen käytetystä ohjelmointikielestä (ks. Kuva 7). Aliohjelmia hyödynnetään kuten lauseita ja niitä voidaan käyttää ohjelman eri osissa. Jos aliohjelma on yleiskäyttöinen laskutoimitus tai usein käytetty tietokoneen toimintojen yhdistelmä voidaan näitä hyödyntää myös muissa ohjelmissa. Aliohjelmia voidaan jakaa muiden käyttäjien kanssa lähdekoodina (ohjelman teksti) tai käännettynä ohjelmana, jos käyttöympäristö on samanlainen.

Ohjelmointikielissä on valmiiksi lauseita peruslaskutoimituksille ja monia muita laskutoimituksia voidaan lisätä erilaisista ohjelmakirjastoista. Näiden hyödyntämiseksi tarvitaan matematiikan osaamista, jotta tiedetään mihin niitä voidaan käyttää. Erityisesti matematiikasta on hyötyä silloin kun oikeaa laskutoimitusta ei löydy valmiina vaan sen algoritmi on muodostettava yksinkertaisimmista laskutoimituksista. Matematiikka on myös avain ongelmanratkaisuun, koska ongelman ratkaiseminen tietokoneella vaatii, että ratkaisu on täsmällinen. Ohjelmointi ja tietokone tarjoaa ongelmanratkaisuun automaatiota, mallintamista, monimutkaisuuden hallintaa, yleistämistä ja tietokoneen oheislaitteiden ominaisuuksia. Ohjelmoinnin ja matematiikan osaaminen tukevat toisiaan ja niiden yhdistäminen tuo esille myös matematiikan hyödyn käytännössä. Esimerkiksi pelien ja animaatioiden toteutuksessa tarvitaan matematiikan kaavoja ruudun tapahtumien kuvaamiseen.

2.2 MATEMAATTINEN ONGELMANRATKAISU

Edellä tuotiin esille, että ohjelmoinnin ja matematiikan yhteys on laskemissa. Matematiikka koostuu eri osa-alueista, jolla jokaisella on omat käsitteet ja niihin perustuvat laskusäännöt. Edistyneemmät osa-alueet rakentuvat yksinkertaisempien osa-alueiden käsitteiden ja sääntöjen pohjalta. Ohjelmointikielillä voidaan kuvata eli koodata matematiikan käsitteitä ja laskusääntöjä. Niiden sisältämät peruslaskutoimitukset antavat tähän lähtökohdan. Tietokone mahdollistaa lisäksi laskutoimitusten automatisoinnin. Ohjelmointikielet ovat teknisiä välineitä, jolloin mahdollisimman pienellä käsitteiden määrällä on haluttu mahdollistaa sekä matemaattisten että teknisten mahdollisuuksien hyödyntäminen. Matematiikan ja tekniikan yhdistäminen vaatii ohjelmoinnissa tarkkuutta.

Matematiikan merkityksen ohjelmoinnissa tuo esille se, että yleisesti käytetyissä ohjelmointikielessä on kaikissa aritmetiikan laskutoimituksia edustavat komennot ja ohjelmointikielten kirjastoista löytyy myös monipuolisempia laskutoimituksia valmiina. Opetuksen kannalta ohjelmointikielet sopivat siksi paremmin matematiikan soveltamisen harjoitteluun. Tyypillisesti soveltavat tehtävät ovat erilaisia ongelmanratkaisutehtäviä. Ohjelmointi tuo näihin uusia mahdollisuuksia kuten tiedon visualisoinnin, matemaattiset mallit, datan käsittelyn ja ohjelmien luomisen. Matemaattinen ongelmanratkaisu on keskeinen ohjelmissa, jotka hyödyntävät paljon laskemista. Siksi matemaattisen ongelmanratkaisun harjoittelu on hyödyllistä ohjelmoinnin kannalta ja siinä opitaan samalla muuttamaan tekstimuodossa oleva kuvaus täsmälliseen ratkaisun mahdollistavaan muotoon.

Koulumatematiikassa ongelmaa voidaan ajatella tehtävänä, jonka ratkaisutapa ei ole oppilaalle ilmeinen (ks. Esimerkki alla). Kyseessä voi olla tapa, jolla ongelma esitetään tai sitten ratkaisu voi vaatia useamman

asian huomioimisen, jolloin etenemistapa ei ole aluksi selvä. Myös haluttu lopputulos voi vaatia pohtimista. Ongelmia ratkaisemalla oppii pärjäämään vastaavanlaisista tehtävistä, mutta samalla oppii yleisiä ongelmanratkaisutapoja. Näihin kuuluu tehtävänannosta olennaisten asioiden poimiminen ja niiden välisten suhteiden hahmottaminen. Myös ohjelmoinnissa haluttu toiminnallisuus kuvataan usein tekstillä. Kuvauksien tarkkuus voi vaihdella täsmällisestä määritelmästä pelkkien pääpiirteiden luettelemiseen. Ohjelman tekijän on kuvauksesta pystyttävä hahmottamaan, millaista ratkaisua haetaan.

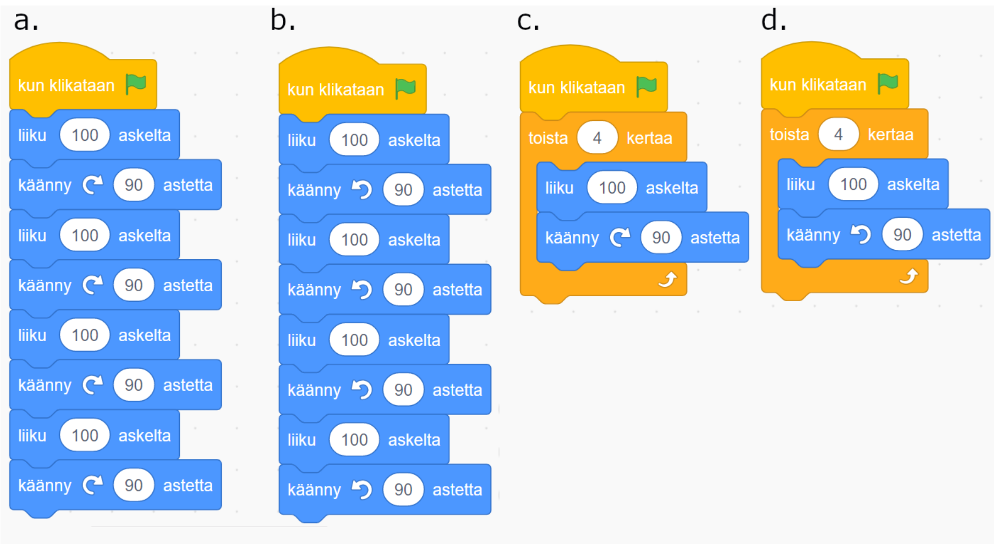
Esimerkki: (muokattu kirjasta Matematiikan tietokirja, 2. painos, Bernoulli ym. 2016)

Tehtävä: Kallella on kaksi kertaa niin monta pehmolelua, kuin Minnalla. Minnalla on kolme pehmolelua enemmän kuin Annalla, jolla on viisi pehmolelua. Kuinka monta pehmolelua on Kallella?

Ratkaisu laskutoimituksella: $2 \cdot (3 + 5) = 2 \cdot 8 = 16$

Osissa ratkaistuna:
Annan pehmolelut: 5
Minnan pehmolelut: Annan pehmolelut + 3 = 5 + 3 = 8
Kallen pehmolelut: 2 * Minnan pehmolelut = $2 \cdot 8 = 16$

Ongelmat voidaan luokitella avoimiin ja suljettuihin. Suljetussa ongelmassa sekä alku- että lopputilanne on määritelty täsmällisesti. Avoimessa ongelmassa alkutilanne, lopputilanne tai molemmat on jätetty määrittelemättä, jolloin niissä voi ongelman ratkaisija tehdä omia valintoja. Jos ongelma on alkutilanteeltaan avoin, ratkaisija voi päättää mistä lähtökohdista päästään annettuun lopputulokseen. Avoimessa lopputilanteessa riittää, että tulos on seuraus lähtökohdista. Molempien ollessa avoimia voi tekijä päättää tehtävän itse, mutta yleensä jotain sen valintaa ohjaavia kriteereitä on kuitenkin annettu. Koulumatematiikassa ongelmat ovat yleensä suljettuja eli oppilaan tulee tietää millainen sääntö tai prosessi tuo ratkaisun. Ohjelmoinnissa lopputilanne on avoin, koska algoritmin toteuttavan ohjelman voi yleensä kirjoittaa monella tavalla (ks. Kuva 8). Alkutilanne on suljettu, koska siinä kerrotaan mitä ohjelman tulisi toteuttaa.



Kuva 8: Ohjelman, joka piirtää neliön, voi kirjoittaa usealla eri tavalla. Tässä on neljä erilaista ohjelmaa Neliö voidaan piirtää neljällä viivalla ja viivojen piirtämisen välissä tekemällä 90 asteen käännöksen. Piirtäminen voidaan tehdä myötä (Kuva 8 a) tai vastapäivään (Kuva 8 b). Ohjelmoinnissa voidaan toistuvat toimenpiteet määritellä toistolauseella, joten riittää että määritellään piirto ja kääntö, joka toistetaan neljästi. Myös tässä voidaan tehdä versiot, joiden piirto etenee joko myötä (Kuva 8 c) tai vastapäivään (Kuva 8 d).

Ongelmanratkaisu on systemaattinen prosessi, josta tunnetuin lienee matemaatikko George Polyan määritelmä. Siinä ongelmanratkaisu on jaettu neljään vaiheeseen: ongelman ymmärtäminen, suunnitelman laatiminen, suunnitelman toteuttaminen ja toteutuksen arviointi. Polyan menetelmä on liian abstrakti koululaisille, joten sitä on hyvä täydentää. Koululaiset olettavat laskutehtävien kokemuksen perusteella, että ongelman kuvauksesta tulisi välittömästi tulla mieleen ratkaisutapa. Ratkaisun keksiminen vaatii kuitenkin usein huolellista ongelman kuvauksen lukemista ja ratkaisun suunnittelua. Suunnitelman toteutus ei välttämättä onnistu. On mahdollista, että kaikkia yksityiskohtia ei tullut huomioitua, vaikka menetelmä sinänsä oli oikea. Valittu menetelmä voi myös olla väärä, jolloin on kokeiltava toista. Oppilaiden olisi hyvä tottua siihen, että epäonnistuminen on osa ongelmanratkaisua ja jokainen epäonnistuminen auttaa ratkaisun löytämisessä. Vaikka on löydetty oikea laskutapa ja saatu tulos, niin ongelma ei ole vielä ratkaistu. On vielä tarkistettava, että tulos on tehtävänannon vaatimassa muodossa esim. koululaisten kevätretken vaatimien linja-autojen määrä ei voi olla desimaaliluku. Ohjelmointi noudattaa samankaltaista ongelmanratkaisuprosessia.

Abstraktioiden hyödyntäminen on yhteistä matematiikalle ja ohjelmoinnille. Matematiikassa esimerkiksi kokonaisluvut edustavat määrää tai järjestystä, mutta todellisuudessa lukuja ei ole, vaan ne kuvaavat tietyn tilanteen keskeisiä piirteitä. Ohjelmoinnissa toimitaan ongelmanratkaisussa samanaikaisesti usealla abstraktiotasolla. Samanlaista näkökulmaa voi hyödyntää myös matematiikassa. Ohjelmoinnissa abstraktiotasot ovat ongelma, algoritmi, ohjelma ja ohjelman suoritus (ks. Taulukko 1 vasen sarake). Matemaattisessa ongelmanratkaisussa näitä vastaa ongelma, ratkaisun kaava ja laskeminen (ks. Taulukko 1 oikea sarake). Ongelmanratkaisuprosessissa liikutaan abstraktiotasolta toiselle, mutta tarvittaessa on osattava siirtyä takaisinpäin, jos tietoa puuttuu tai aikaisemmin tehdyt päätökset eivät toimi.

Abstraktiotasojen sekoittaminen aiheuttaa ongelmia sekä ohjelmoinnissa, että matematiikassa. Oppilaita voidaan auttaa abstraktiotasojen hyödyntämisessä opettamalla vaiheittaista etenemistä ongelmanratkaisussa ja harjoitella palaamista edelliseen vaiheeseen tarvittaessa.

Taulukko 1: Ongelmanratkaisun abstraktiotasot ohjelmoinnissa ja matematiikassa.

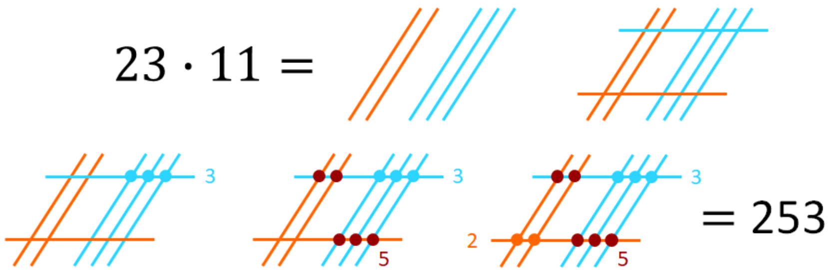
Ohjelmoinnin abstraktiotasot	**Matematiikan abstraktiotasot**
Ongelma	Ongelma
Algoritmi	Matemaattinen kaava
Ohjelma	Laskeminen
Ohjelman suoritus (tietokone)	-

Matematiikan ja ohjelmoinnin ongelmanratkaisussa on yhtäläisyyksiä ja eroavaisuuksia. Matematiikka on osa ohjelmointia ja samalla ohjelmointi tarjoaa mahdollisuuden soveltaa matematiikkaa. Yläkoulun tavoitteissa ongelmanratkaisutavat yhdistetään eksplisiittisesti, mutta alakoulussa voidaan pohjustaa tätä harjoittelemalla näitä yhdessä ja erikseen. Ratkaisumenetelmän vaiheet ovat yhteiset: ongelman ymmärtäminen, suunnitelman, toteutus ja arviointi. Kun tarkastelemme matematiikan ja ohjelmoinnin abstraktiotasoja huomaamme kuitenkin eron. Matematiikassa ratkaisua edustaa kaava, jonka oppilas laskee. Ohjelmoinnissa ratkaisu on algoritmi, jonka toteuttava ohjelma kirjoitetaan ja lopuksi suoritetaan. Algoritmit kuuluvat myös matemaattiseen ongelmanratkaisuun, jos ongelma vaatii askelittaisen käsittelyn.

2.3 ALGORITMINEN ONGELMANRATKAISU

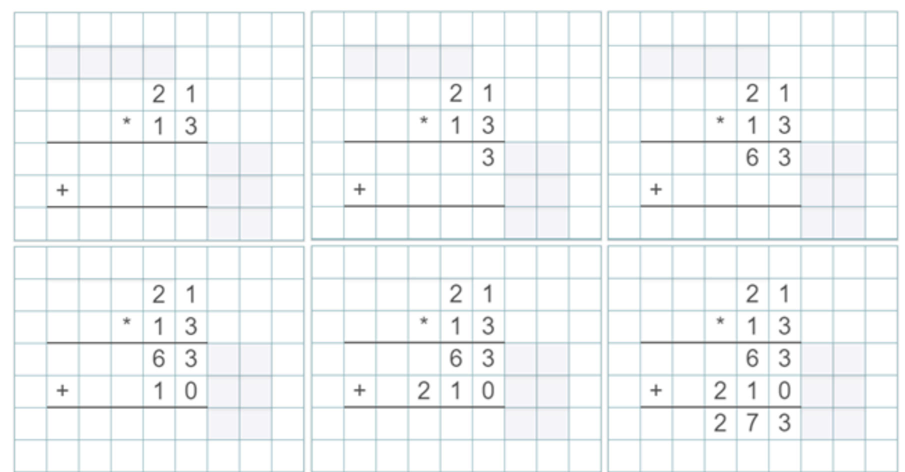
Ongelmanratkaisu on yhteistä matematiikalle ja ohjelmoinnille, mutta matematiikassa ratkaisu perustuu kaavoihin ja ohjelmoinnissa algoritmeihin. Algoritmit keksittiin kuitenkin matematiikassa ja niiden avulla voidaan määritellä laskutoimitus yksiselitteisesti. Alun perin algoritmien tarkoitus oli mahdollistaa monimutkaisten laskutoimitusten suorittaminen yksinkertaisia yhdistelemällä. Ihminen voi turvautua uuden oppimisessa aikaisempaan osaamiseensa, joten siksi laskutoimitukset esitellään kaavoina ja mainitaan vain se asia mikä on uutta. Proseduraalisessa osaamisessa voisi yhtä hyvin puhua algoritmisesta, koska matematiikassa proseduurien on oltava täsmällisiä. Algoritmit voidaan esittää peräkkäis-, toisto- ja ehtorakenteella sekä näiden sisältämillä laskutoimituksilla. Tässä tutustutaan tarkemmin näihin kolmeen rakenteeseen ja niiden käyttöön ongelmanratkaisussa.

Jokainen algoritmi noudattaa peräkkäisrakennetta ja askelia on oltava kaksi tai useampia (ks. Kuva 9). Yksi askel edustaisi vain laskutoimitusta, joten se ei kelpaa algoritmiksi. Askeleet voivat sisältää laskutoimituksia tai muita algoritmin rakenteita. Ne voivat toistaa aikaisemmin tehtyjä asioita tai tehdä uusia. Algoritmin loppupuolella usein kootaan aikaisemmin saatuja tuloksia.



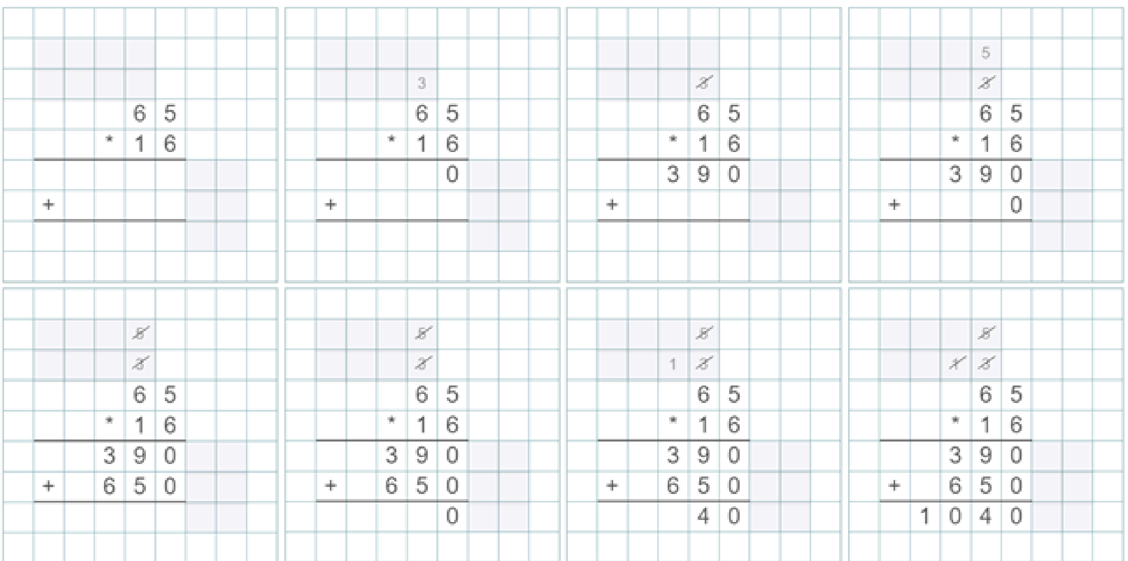
Kuva 9: Visuaalinen kertolaskualgoritmi, jossa sininen viiva edustaa lukua yksi ja oranssi lukua kymmenen. Luvut muodostetaan sinisten ja oranssien viivojen yhdistelmistä. kertolaskua varten ensimmäinen luku esitetään pystyviivoilla ja toinen luku vaakaviivoilla. Tulos saadaan laskemalla viivojen risteyskohtia huomioiden risteävien viivojen yksiköt. Algoritmi etenee suoraviivaisesti, ensin esitetään ensimmäinen luku ja toinen luku sen päälle vaakasuoraan, jonka jälkeen lasketaan ykköset, joita edustaa risteävät siniset viivat, sitten kymmenet, joissa sininen ja oranssi viiva risteää ja lopuksi sadat, joissa kaksi oranssia viivaa risteää.

Toistorakenteessa suoritetaan uudelleen jotain peräkkäisrakenteen askelten joukkoa, kunnes jokin ehto (vrt. ehtorakenne alla) täyttyy (ks. Kuva 10). Tämä säästää kirjoittamasta samanlaista toimenpidesarjaa useaan otteeseen. Valitsemalla toiston päättymisehdot sopivasti, esim. niin kauan kuin käsiteltäviä lukuja löytyy tai kunnes joku arvo kasvaa tietyn suuruiseksi, voidaan luoda yleiskäyttöisiä algoritmeja.



Kuva 10: Kahden kaksinumeroisen luvun kertolasku allekkain on esimerkki algoritmista, jossa hyödynnetään toistoa. Kerrotaan ensin ensimmäinen luku toisen luvun vähemmän merkitsevällä ja sitten merkitsevämmällä numerolla. Molemmissa tapauksissa edetään vastaavasti ja tässä esimerkissä laskutoimitukset eivät vaadi muistinumeroa. Tuloksena saadaan kaksi lukua, jotka lasketaan lopuksi yhteen.

Ehtorakenne mahdollistaa ehtojen asettamisen käsiteltäville arvoille, jolloin niiden perusteella voidaan vaikuttaa algoritmin etenemiseen (ks. Kuva 11). Tyypillisesti on kaksi vaihtoehtoa: tehdään joku toimenpide ehdon täyttyessä tai siirrytään eteenpäin. Ehtorakenne voi liittyä yhteen toimenpiteeseen, valintaan kahden toimenpiteen välillä tai se voi jakaa algoritmin kahteen rinnakkaiseen haaraan.



Kuva 11: Kun kahden kaksinumeroisen luvun kertolaskussa allekkain numerot ovat riittävän isoja, on myös käsiteltävä tilanne, jossa syntyy nk. muistinumeroita. Jos kahden luvun allekkain kertolaskussa tulos on yli kymmenen, merkitään kymmenet seuraavan numeron päälle muistinumerona tai jos numerot loppuvat niin luvun eteen. Kun seuraavan numeron laskutoimitus on suoritettu, lisätään vielä muistinumero. Muistinumeron käyttöä voidaan pitää ehtona, joka tulee voimaan vain, kun kertolaskun tuloksena syntyy kymmenlukuja.

Algoritmin rakenteet ovat yksinkertaisia, ja kun osataan niiden sisältämien askeleiden laskutoimitukset, on algoritmin noudattaminen suoraviivaista. Myös matematiikassa voidaan käyttää algoritmeja ongelmanratkaisussa, kun ongelma vaatii askelittaisen etenemisen (ks. Kuva 12).

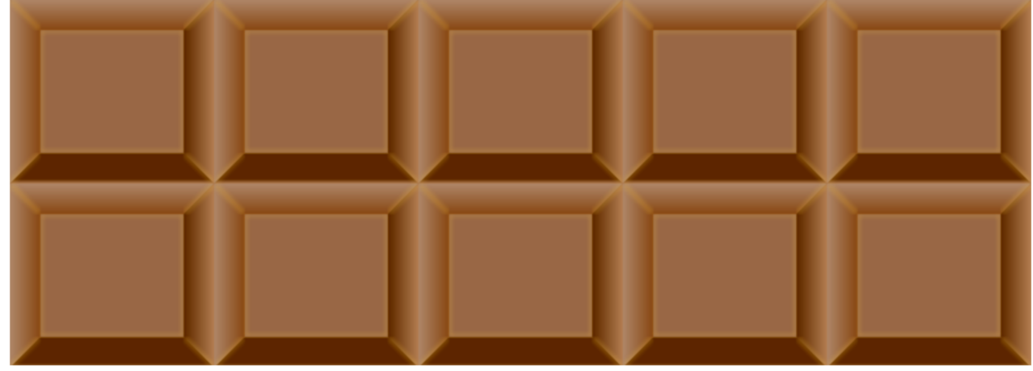


Image: Lord Belbury (CC BY-SA 4.0) https://commons.wikimedia.org/wiki/File:Chomp_game.png a part of the original image.

Kuva 12: Tehtävä: Suklaalevyä halkoo pysty- ja vaakasuuntaiset uurteet, jotka muodostavat kymmenen palaa. Kuinka monella leikkauksella suklaalevy voidaan jakaa palasiin uurteiden mukaan, jos jokainen leikkaus aina jakaa levyn tai myöhemmin sen osia kahtia? Vastaus: Suklaalevyn pilkkominen osiin vaatii 9 leikkausta. Algoritmin askeleessa 0, osia on 1 ja leikkauksia on 0. Askeleessa 1, osia on 2 ja leikkauksia 1. Askeleessa 2, osia on 3 ja leikkauksia 2. Askeleessa 3... Osia on aina yksi enemmän kuin leikkauksia, joten jos levyssä on 10 palaa, niin niiden erottaminen vaatii 9 leikkausta.

Suunniteltaessa algoritmia voi myös huomata, että jokaisessa algoritmin askeleessa jokin arvo tai useamman arvon välinen suhde noudattaa samaa sääntöä. Näitä kutsutaan invarianteiksi (alkuperäinen englanninkielinen sana tarkoittaa muuttumatonta tai pysyvää). Algoritmin voidaan ajatella toteuttavan invariantin ja kun varmistetaan, että oikeanlaisella syötteellä algoritmi päättyy aina, ollaan jo pitkällä algoritmin suunnittelussa. Suklaalevyn esimerkissä osien ja leikkausten suhde pysyy samana, jolloin osia on yksi enemmän kuin tehtyjä leikkauksia. Ongelmaa voidaan ajatella syötteenä algoritmille ja lopputulosta sen suorittamisen seurauksena. Ongelman ratkaisemiseksi algoritmisesti on keksittävä, millaisin askelin päädytään syötteestä tulokseen. Algoritmia suunniteltaessa voidaan keksiä myös sitä kuvaava kaava, jolloin ratkaisu voidaan tehdä laskemalla. Suklaalevyn jakamisen osiin tapauksessa leikkausten määrä saadaan vähentämällä palojen määrästä 1.

Algoritmisessa ongelmanratkaisussa voidaan käyttää apuna kolmea strategiaa: ongelman lähtöarvojen ja lopputuloksen määrittely, asteittainen tarkennus ja ratkaisumallit. Ensin on määriteltävä mitkä ovat lähtöarvot, jotka toimivat algoritmin syötteenä, ja lopputulos, johon pyritään. Kun tiedetään syötteet ja tulos voidaan jakaa ongelma loogisesti osaongelmiin asteittain tarkentuvina askelina. Ongelman pilkkomisessa on pidettävä ennallaan syötteen ja tuloksen välinen suhde. Pilkkomisessa hyödynnetään algoritmin perusrakenteita ja laskutoimituksia. Pilkkomista jatketaan, kunnes on saavutettu taso, jossa kaikki operaatiot ovat atomisia (niitä ei voi enää pilkkoa osiin). Ratkaisumalleja käytetään osana oman algoritmin muodostamista. Tunnettujen mallien hyödyntämisen etuna on, että tiedetään niiden toimivan oikein. Matematiikassa erilaisten kaavojen hyödyntäminen edustaa tällaisia malleja.

Ohjelmoinnissa algoritmit toimivat ohjelman osana, mutta ne eivät vielä riitä ohjelman muodostamiseen. Algoritmit on määritelty ihmistä varten ja niiden suorittaminen tietokoneella vaatii niiden kuvaamisen ohjelmointikielellä (ks. Taulukko 2). Lisäksi on ohjattava tietokonetta sekä noudatettava ohjelmointikielen

määrittelemiä yleisiä käytänteitä. Siksi tarkastikin määritelty algoritmi vaatii vielä ohjelmointityötä. Usein algoritmi on osa isompaa kokonaisuutta, joten nämä piirteet on myös kuvattava tai jos ne ovat valmiina, niin sovitettava algoritmi niihin. Alakoulun ohjelmoinnissa näitä eri osa-alueita voidaan harjoitella erikseen, mutta kun halutaan luoda monipuolisempia ohjelmia kuten pelejä ja animaatioita on myös kokonaisuuden muodostamista harjoiteltava.

Taulukko 2: Algoritmin ja ohjelman erot

	Algoritmi	**Ohjelma**
Kohde	Henkilö	Tietokone
Kieli	Yleensä esitetään epäformaalilla kielellä	Esitetään ohjelmointikielellä
Yksityiskohtaisuus	Epäolennaisia yksityiskohtia ei mainita	Kaikki yksityiskohdat määritellään
Tarkkuus	Riittävän tarkka, jotta yleistiedolla ja riittävällä taustoituksella, henkilö voi sen toteuttaa.	On määriteltävä tarkkuudella, joka ei jätä mitään arvailun varaan.

2.4 OHJELMOINNILLINEN AJATTELU ONGELMANRATKAISUSSA

Edellä on esitetty, että matematiikan ja ohjelmoinnin yhteys on laskemisessa. Tarkemmin tämä yhteys on algoritmeissa, joiden avulla laskeminen toteutetaan ohjelmoinnissa. Ohjelmoinnissa algoritmit nousevat esille, koska toteutamme niitä ja ohjelmien rakenne perustuu niihin. Edellä esitettiin miten peräkkäis-, toisto- ja ehtolause yhdistettynä muuttujien käsittelyyn muodostavat algoritmin ytimen. Ohjelmoinnin taustalla on ajattelumalli, jossa em. käsitteitä hyödynnetään ongelmanratkaisussa. Käsitteiden lisäksi ohjelmoinnillinen ajattelu sisältää joukon älyllisiä taitoja, joita tarvitaan käsitteiden hyödyntämisessä. Näihin kuuluu abstrahoinnin, loogisen päättelyn, data-analyysin, ongelman analysoinnin, algoritmisen ajattelun, yleistämisen ja automaation taidot.

Abstrahointi on kohteena olevan asian keskeisten piirteiden tunnistamista ja ylimääräisten tai tarpeettomien piirteiden jättämistä pois. Liian paljon tietoa häiritsee ongelman käsittelyä, koska ihmisen työmuisti on rajallinen (sanotaan että työmuistissa voi käsitellä noin seitsemää asiaa samanaikaisesti). Erityisesti ohjelmointiin liittyen abstrahoinnissa voidaan antaa nimi joukolle asioita ja näin käsitellä joukon abstraktiota (substantiivi). Esimerkiksi aliohjelmat tulisi nimetä kuvaavasti, jotta ei tarvitsisi ajatella lauseiden joukkoa vaan niiden merkitystä.

Looginen päättely on johdonmukaista ajattelua, joka auttaa löytämään järkeviä yhteyksiä asioiden välillä sekä auttaa tosiasioiden todentamisessa ja tarkistamisessa. Algoritmien muodostaminen on seurausta

loogisesta päättelystä, jossa lähtökohdasta (syöte) johdetaan toimenpiteiden ketju, joka johtaa lopputulokseen (tulos). Kun algoritmi toteutetaan ohjelmoimalla, on pidettävä huolta, että sen logiikka säilyy, kun se kuvataan ohjelmointikielen käsitteillä. Päättelyä tarvitaan myös ongelman paikantamiseen, kun ohjelma ei tee toivottua asiaa tai tekee sen väärin.

Data-analyysi on hyödyllisen informaation löytämistä suuresta määrästä dataa. Sen avulla voidaan löytää säännönmukaisuuksia ja tehdä niihin perustuvia oivalluksia. Säännönmukaisuudet vihjaavat miten dataa tulisi käsitellä, jotta siitä voisi jalostaa informaatiota. Erilaiset internetin palvelut sekä käyttämämme digitaaliset laitteet keräävät jatkuvasti dataa. On keskeistä ymmärtää, mitä erilaiset datalähteet kertovat ja miten niiden tuottama data voidaan yhdistää merkityksen luomiseksi. Ohjelmoinnissa on tunnistettava datan rakenne, jotta sitä voisi käsitellä oikein. On tärkeä erottaa, millaisessa muodossa data tallennetaan ja miten se saadaan palautettua muotoon, jossa sitä on kätevä käsitellä.

Ongelman analysointi on tiedon tai ongelman jakamista pienempiin ja paremmin hallittaviin osiin. Kun ongelma jaetaan osiin, saadaan esille osien piirteet ja voidaan päätellä niiden merkitys kokonaisuudessa. Toinen tärkeä tekijä on osien väliset suhteet, miten eri osien piirteet ovat vuorovaikutuksessa toistensa kanssa. Matematiikan osa-alueet tutkivat erilaisia käsitteitä ja niiden välisiä suhteita. Näiden perusteella voidaan laskea erilaisia tuloksia niiden mukaisista kohteista. Edellä esitetty data-analyysi voidaan ajatella ongelmana, jossa data on jaettava osatekijöihin, sen sisältämän rakenteen esille tuomiseksi.

Algoritminen ajattelu on jonkin ongelman ratkaisemiseksi tai tehtävän suorittamiseksi tarvittavien toimenpiteiden ja niiden suoritusjärjestyksen määrittämistä. Olemme edellisissä osissa esittäneet algoritmit matematiikkaa ja ohjelmointia yhdistävänä tekijänä. Kummassakin hyödynnetään algoritmista ajattelua, mutta koska matematiikassa laskija on ihminen, voidaan olla summittaisempia, kuin jos ohjataan digitaalista konetta. Algoritmisen ajattelun kehittäminen on yksi yläkoulun matematiikan yhteydessä opetettavan ohjelmoinnin tavoitteista.

Järjestelmällisessä arvioinnissa huomioidaan kaikki tekijät, jotka vaikuttavat siihen, miten ajattelemme ja mitä päätämme tehdä. Tilanteen tai tehtävän järjestelmällinen arviointi auttaa löytämään parhaan mahdollisen ratkaisun. Usein ohjelmoinnissa ei ole selkeää määritelmää mistä aloittaa, vaan kyse on ongelmallisesta tilanteesta, johon kaivataan ohjelmallista ratkaisua. Silloin tilanteen järjestelmällinen arviointi auttaa tuomaan esille vaihtoehdot ja valitsemaan niistä ne, jotka parhaiten voisivat ratkaista ongelman. Järjestelmällisyys on keskeistä, koska ongelmallisille tilanteille on luontaista, että ne ovat epämääräisiä ja siksi vaikeasti hallittavia.

Yleistäminen auttaa ennakoimaan asioiden seurauksia luomalla malleja, sääntöjä, periaatteita tai teorioita havaituista säännönmukaisuuksista. Kun tiettyyn ongelmaan on kehitetty hyvä ratkaisu, voidaan miettiä toimisiko ratkaisu laajemminkin. Tämä on keskeinen osa ohjelmoinnin kehittämistä. Epävirallisemmin, ohjelmoijat hyödyntävät mielellään tuntemiaan ratkaisuja uusien sijaan. On tutkittu, että jos ohjelmoijat tutustuvat kirjastoon ratkaisumalleja ennen ohjelman kehitystä, he ottavat paljon todennäköisemmin niitä käyttöön. Ohjelmistotekniikassa kehitetään yleisiä ratkaisumalleja tyypillisin

ongelmiin ja tietojenkäsittelytieteessä tutkitaan hyödyllisiä algoritmeja. Nykyään Internetistä voi helposti löytää valmiita ratkaisuja, mutta on osattava arvioida sopivatko ne juuri omaan käyttötarkoitukseen.

Automaatio tarkoittaa ohjelmoinnillisessa ajattelussa ongelman saattamista sellaiseen muotoon, että sen ratkaisun voisi suorittaa automaattisesti. Tämä tarkoittaa ratkaisun ohjelmoimista ja sen suorittamista tietokoneella. On hyvä tunnistaa, milloin ratkaisu on järkevä toteuttaa automatisoimalla. Esimerkiksi helppoa päässälaskutehtävää ei kannata suorittaa tietokoneella. Automaatio on hyödyllistä silloin, kun ongelman monimutkaisuus tai datan määrä on niin suuri, että ratkaisun ohjelmoinnin vaiva kannattaa. Toinen syy, on tarve tietokoneen tai sen lisälaitteiden ominaisuuksille, joita ei muuten saataisi käyttöön.

Ohjelmoinnillinen ajattelu on ongelmanratkaisumenetelmä, joka hyödyntää edellä mainittuja taitoja. Taitoja voidaan hyödyntää ongelmanratkaisun kaikissa vaiheissa, mutta tietyissä vaiheissa ne ovat keskeisiä (ks. Taulukko 3). Ongelman ymmärtämisessä on hyvä edetä järjestelmällisesti, sen mahdollisia tulkintoja arvioiden. Kun ongelma on tunnistettu, on valittava keskeiset tekijät abstrahoimalla ja sitten tutkia niitä jakamalla ongelma osatekijöihin. Ratkaisun suunnittelu tapahtuu saatavilla olevaa dataa analysoimalla ja käyttämällä algoritmista ajattelua datan hyödyntämisen suunnitteluun. Toteutuksessa käytetään loogista päättelyä ja ohjelmointikielen käsitteitä algoritmin suorituksen automatisointiin. Lopuksi voidaan arvioida toteutusta ja havaita mahdollisuuksia ratkaisun käyttämiseen yleisemmin.

Taulukko 3: Ohjelmoinnillinen ajattelun kykyjen sijoittuminen ongelmanratkaisun vaiheisiin.

Ongelman ymmärtäminen	**Suunnitelman laatiminen**	**Suunnitelman toteuttaminen**	**Toteutuksen arviointi**
Abstrahointi Ongelman analysointi Järjestelmällinen arviointi	Algoritminen ajattelu Data-analyysi	Automaatio Looginen päättely	Yleistäminen

Ohjelmoinnillista ajattelua pyritään opettamaan ohjelmoinnin harjoituksilla. Ilman ajattelua ohjelmointi perustuu yritys/erehdys -menetelmään ja monimutkaisempien ohjelmien luonti on mahdotonta. Ohjelmat voivat silloin olla kopioita tai yhdistelmiä aikaisemmista ohjelmista ilman suunnitelmaa. Ne voivat tehdä sen mitä pyydetään, mutta ne voivat myös sisältää paljon turhaa ja tehdä asioita, joita ei ole pyydetty. Ohjelmoinnin harjoittelu ei ole silloin johtanut ymmärtämiseen. Ohjelmoinnillisen ajattelun malli auttaa opettajaa tukemaan oppilaan kehittymistä ohjelmoinnissa. Se mahdollistaa tarvittavien taitojen harjoittelua jo ennen kuin siirrytään tietokoneen ääreen. Ohjelmoinnin ymmärtäminen tukee aktiivista osallistumista digitaalisen yhteiskuntaan ja perustaitojen osaaminen avaa uusia opiskelu- ja uramahdollisuuksia.

3 TYÖPAJA 1: OHJELMOINNILLISEN AJATTELUN HARJOITUKSIA ALKUOPETUKSEEN

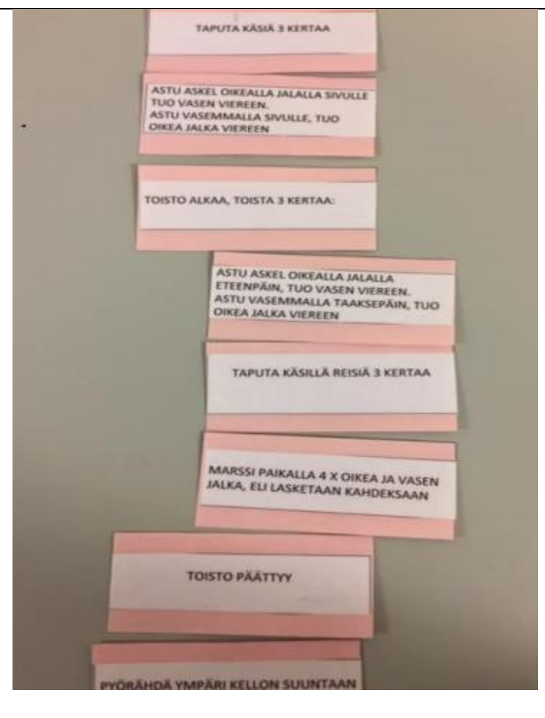
Työpajan tavoitteena oli esitellä erilaisia ”unplugged” eli ilman tietokonetta tehtäviä ohjelmoinnillisen ajattelun harjoituksia. Harjoitusten aiheet liittyvät abstrahoinnin, loogisen päättelyn, data-analyysin, ongelman analysoinnin, algoritmisen ajattelun, yleistämisen ja automaation taitoihin. Myös tietokoneisiin ja tietotekniikkaan liittyvää tietoutta voidaan opettaa. Unplugged -lähestymistavan etuna on, että voidaan keskittyä juuri harjoitettavaan asiaan, ilman että käytettävä teknologia vie huomion. Harjoitukset voivat olla kehollisia, niitä voidaan tehdä perinteisesti paperilla ja kynällä tai käyttäen jotain apuvälinettä kuten pelikortteja. Tietokoneresurssit ovat usein koulussa rajalliset, joten unplugged -harjoituksilla voidaan säästää näitä ja olla paremmin valmistautuneita, kun tietokoneet ovat käytössä. Unplugged -harjoitusten painopiste on alkuopetuksessa, mutta myös myöhemmin voidaan uusi tai hankala asia selkeyttää näillä.

Työpajassa oli lyhyen alustuksen jälkeen kolme erillistä osiota, jossa harjoiteltiin toimintaohjeiden noudattamista tanssien, ongelmanratkaisua liikkuen ja paperilla, sekä ohjelmoinnillisen ajattelun ja tietotekniikka tietouden lisäämistä Majava-korttien tehtäviä ratkomalla.

Tanssikoodaus

<table>
<tr>
<td>

Tavoitteena on toteuttaa tanssi annettuja ohjeita täsmällisesti seuraamalla
Tanssin koodaaminen tapahtuu noudattamalla korttipakan ohjeita, jonka kortit sisältävät tanssiliikkeitä tai muita ohjeita (esim. toistoja)
Korttipakka sekoitetaan, joten ohjeet tulevat satunnaisessa järjestyksessä
Tanssia tukemaan voidaan käyttää sopivan rytmillistä musiikkia
Laatinut: Anu Tuominen/OKL/Turun yliopisto

</td>
<td>


</td>
</tr>
</table>

Ongelmanratkaisu CS unplugged

• Toimintaohjeiden toteuttaminen, jossa toinen toimii robottina ja toinen antaa täsmällisiä ohjeita • Kuvien esittäminen digitaalisessa muodossa, jossa kuva muodostuu pisteistä koordinaatistossa • Kuvien koodaaminen binäärimuotoon ja miten värien lisääminen vaikuttaa esitykseen • LINKKEJÄ CS unplugged https://csunplugged.org/en/ https://csunplugged.org/en/at-home/squeezing-pictures/ • London computing https://teachinglondoncomputing.org/	 Image: The CS Unplugged is a project by the Computer Science Education Research Group at the University of Canterbury, New Zealand. (CC BY-SA 4.0) https://www.csunplugged.org/static/img/topics/unplugged-programming-icon.png

Majava-kortit

• Bebras-kilpailu Liettuasta, jossa osallistujia ympärimaailmaa (bebras.org) • Opettaa hauskalla ja motivoivalla tavalla tietojenkäsittelytieteen ja ohjelmoinnillisen ajattelun peruskäsitteitä. • Harjoittelu pelikorteilla ja digitaalisilla tehtävillä. • Suomalainen Majava-kilpailu järjestetään vuosittain (majava-kilpailu.fi)	 Kuva: muokattu, osoitteesta majava-kilpailu.fi.

4 VERKKOSISÄLTÖ 2: TIETOKONEEN MAHDOLLISUUKSISTA OHJELMOINTIIN

Aiemmissa osioissa on käsitelty ohjelmointia ja matematiikkaa, algoritmeja ja ohjelmoinnillista ajattelua. Tavoitteena oli tuoda esille, miten ohjelmointi ja matematiikka liittyivät toisiinsa. Yhteistä oli ongelmanratkaisu, jossa matematiikalla on tärkeä osa ratkaisun suunnittelussa ohjelmoimalla. Ohjelmoinnillinen ajattelu nosti esille yleisiä taitoja, joita tarvitaan ohjelmoinnissa. Edellisessä työpajassa esiteltiin erilaisia menetelmiä, joilla näitä taitoja voidaan harjoitella ilman tietokonetta. Nyt siirrytään tietokoneen mahdollisuuksien hyödyntämiseen ja johdatellaan ohjelmointiin.

Tämän osion aiheena on tietokone ohjelmointivälineenä. Ensiksi käsitellään ohjelman merkitystä tietokonelaitteen toiminnan osana ja mitä ohjelmointi on tästä näkökulmasta. Ohjelmoinnin taitoja voidaan harjoitella ilman tietokonetta, mutta tietokonelaitteiden yleisyys (pöytätietokoneet, kannettavat tietokoneet, tabletit, älypuhelimet jne.) tekee luontaiseksi näiden käyttämiseen ohjelmoinnillisen ajattelun oppimiseen. Seuraavaksi esitellään miten erilaisilla ohjelmilla ja peleillä voidaan harjoitella ohjelmoinnillisen ajattelun taitoja ennen varsinaista ohjelmointia.

Graafiset ohjelmointiympäristöt madaltavat ohjelmoinnin oppimisen kynnystä, koska niissä voidaan ohjelmoida hyödyntämällä visuaalista komentopalettia. Näin ei tarvitse muistaa komentoja ulkoa ja käytön helppous tukee kokeilemista. Graafinen ohjelmointi on saavuttanut suuren suosion ja useita ohjelmointiympäristöjä on kehitetty tätä varten. Erilaisia ohjelmointiympäristöjä esitellään verkkoluennon neljännessä osassa. Ohjelmointikieliä voidaan luokitella niiden keskeisten periaatteiden perusteella ja verkkoluennon lopuksi esitetään suosituimman nk. imperatiivisen ohjelmointikielten periaatteita edustavat käsitteet. Imperatiivisten ohjelmointikielten keskeiset käsitteet ovat komentoja tai käskyjä, joilla ohjataan tietokoneen toimintaa. Kurssilla käytetty graafinen ohjelmointikieli Scratch voidaan luokitella kuuluvaksi imperatiivisiin ohjelmointikieliin.

4.1 TIETOKONE JA OHJELMOINTI

Tietokoneet, on sitten kyse pöytäkoneesta, kannettavasta, tabletista tai älypuhelimesta, pystyvät esittämään monenlaisia asioita. Ruudulla näkyy mm. tekstiä, numeroita, ikoneja, graafisia elementtejä, kuvia ja videoita. Laitteet pystyvät toistamaan ääntä musiikista puheeseen. Tulostimella voidaan siirtää ruudulla näkyviä asioita fyysiseen muotoon paperille ja käyttäen erityistä tulostinta jopa luoda kolmiulotteisia esineitä. Voimme kirjoittaa tietokoneen fyysisellä tai virtuaalisella (näyttö-) näppäimistöllä, jolloin ruudulla näkyvät ne merkit, jotka vastaavat painettuja näppäimiä. Hiirellä voimme manipuloida ja liikuttaa asioita ruudulla. Skannerilla siirtyvät valokuvat ja muut printtimedian tuotteet tietokoneelle. Nykyään voidaan digitaalisilla kameroilla ottaa valokuvia ja videoita, jotka ovat suoraan käytettävissä tietokoneella. Voimme piirtää hiirellä, sormella tai käyttäen piirtoalustaa. Mikrofonilla voimme tallentaa ääntä ja sopivilla ohjelmilla voimme luoda sitä myös keinotekoisesti. Kuitenkin, jos voisimme tarkastella tietokoneen sisintä, niin emme löytäisi mitään mainituista asioista.

Mitä voimme havaita tietokoneesta on sen tulostetta (engl. *output*), joka on seurausta siihen kytkettyjen laitteiden toiminnasta. Myös syötteet (engl. *input*) vaativat omat laitteensa. Tulosteesta ja syötteestä vastaavat laitteet muuttavat digitaalisen fyysiseksi tai toisinpäin. Digitaalisuudella viitataan numeromerkkiin, joka voi olla nolla tai yksi. Yksi tällainen merkki edustaa binäärilukua ja käyttämällä useampaa lukua voidaan määritellä kaikki tulosteet ja syötteet. Ulkoisia laitteita ohjaa tietokoneen keskusyksikkö, joka on kytketty komponentteihin, jotka ovat varsinaisesti laitteiden kanssa yhteydessä. Binääriluvut esitetään tietokoneessa sähköisesti ja näin ne voidaan välittää johdoilla keskusyksikön ja komponenttien välillä. Ohjaaminen tapahtuu numeroita käsittelemällä eli laskemalla. Matematiikassa keksittiin binääriluvuilla laskevan koneen idea ensin, mutta tietokoneen synty on seurausta laskentaan käytettävien laitteiden omasta kehityksestä.

Alan Turing keksi vuonna 1936 kuvitteellisen koneen, jolla voitiin määritellä kaikki laskutoimitukset yksiselitteisesti ilman päässälaskuun perustuvia välivaiheita. Turing otti koneen lähtökohdaksi laskemisen ruutupaperille, jota hän yksinkertaisti, kunnes laskenta oli mahdollista tehdä täysin mekaanisesti. Paperi voitiin korvata riittävän pitkällä (äärettömällä) ruutuihin jaetulla nauhalla. Jokainen ruutu sisälsi yhden merkin ja kone laski ruudun kerrallaan. Ihminen voi luottaa tehtävän ymmärtämiseen, mutta kone tarvitsi yksiselitteiset säännöt. Sääntöjen suorittaminen alkoi ensimmäisestä säännöstä ja ensimmäisestä ruudusta. Jokainen sääntö viittaasi myös seuraavaan sääntöön, paitsi sääntö, joka määrittelee laskutoimituksen valmiiksi. Sääntöjen valintaan vaikutti myös kohdalla olevan ruudun sisältö ja eri sisältöjä varten oli säännöistä omat versiot. Numeroiksi valittiin binääriluvut, koska niiden käsittelyyn tarvittiin vähiten sääntöjä. Laskemisen yksinkertaistamisesta syntyi kuvitteellinen äärettömällä nauhalla laskeva kone, nk. Turingin kone (TK; ks. Kuva 13).

Säännöt

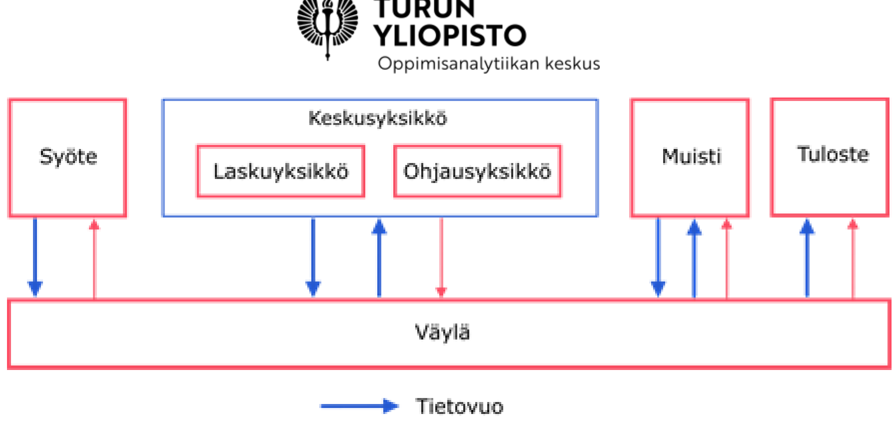
Tila	Luettu merkki	Kirjoitettava merkki	Siirtyminen Vas/Oik/Paik	Seuraava tila
T_1	0	0	*Oik*	T_1
T_1	1	1	*Oik*	T_2
T_1	*Loppu*	*Ei*	*Paik*	*Pysähdy*
T_2	0	0	*Oik*	T_2
T_2	1	1	*Oik*	T_3
T_2	*Loppu*	*Pariton*	*Paik*	*Pysähdy*
T_3	0	0	*Oik*	T_3
T_3	1	1	*Oik*	T_2
T_3	*Loppu*	*Parillinen*	*Paik*	*Pysähdy*

Kuva 13: Alan Turingin (1936) keksimä kuvitteellinen tietokone, joka voi toteuttaa kaikki laskettavissa olevat laskutoimitukset. Kuvitteellinen tietokone koostuu äärettömästä nauhasta, joka on jaettu ruutuihin ja jokainen ruutu voi sisältää yhden merkin. Tässä esimerkissä käytetään 0, 1 Loppu, Pariton ja Parillinen merkkejä. Koneessa on lukupää, joka lukee, kirjoittaa ja liikkuu nauhalla. Koneen toiminnan määrittelevät säännöt, jotka ohjaavat lukupäätä. Tässä sääntöjen määrittelemä ohjelma laskee, onko numeroa yksi parillinen vai pariton määrä, ja kun se kohtaa nauhalla merkin "Loppu", kirjoittaa se vastauksen merkin päälle.

Turingin kone (TK) oli matemaattinen malli, jolla voitiin tutkia, mitä pystytään laskemaan ja mitä ei. Tietojenkäsittelytieteessä havaittiin sen olevan myös osuva määritelmä ohjelmointikielille. Sanotaan että ohjelmointikieli on yhtä ilmaisukykyinen kuin TK, jos sillä voidaan määritellä samat operaatiot. Moderneissa ohjelmointikielissä on monia muita komentoja, jotka tekevät ohjelmoinnin kätevämmäksi, mutta ne rakentuvat pienestä joukosta komentoja (vrt. aliohjelmat), jolla manipuloidaan nollien ja ykkösten sarjoja. TK:lla pystytään koodaamaan minkä tahansa matematiikan osa-alueen laskusäännöt ja vastaavasti myös ohjelmointikielillä. Täsmällisten nk. formaalien kielten määrittely on yksi matematiikan osa-alue. Jotta voimme ohjata tietokonetta on ohjelmointikielellä kirjoitettu ohjelma vielä muutettava eli käännettävä muotoon, joka määrittelee tietokoneen toiminnan.

Turingin kone oli liian abstrakti tietokoneen malliksi, vaikka periaatteessa se toimii vastaavasti. Tietokone on seurausta laskevien laitteiden kehityksestä, jossa rajattuun matematiikan osa-alueeseen keskittyvistä laskimista kehittyi kaikkiin laskutoimituksiin pystyviä tietokoneita. Erityistä oli myös mahdollisuus tallentaa osatuloksia, jolloin voitiin määritellä monimutkaisia laskutoimituksia yhdellä kerralla. Maailman ensimmäisenä tietokoneena pidetään ENIACia (Electronic Numerical Integrator and Computer). ENIAC oli modulaarinen elektroninen tietokone, joka koostui erillisistä yksiköistä. Kaksikymmentä näistä oli laskimia, jotka pystyivät yhteen- ja vähennyslaskuun. Lukujen esitys vastasi kymmenjärjestelmää. Luvut syötettiin laskimien kytkimillä ja johdoilla siirrettiin välituloksia yksiköltä toiselle. Näin voitiin laskea useampia välivaiheita, mutta laskimien määrä toimi rajana.

ENIACissa ohjelma määriteltiin fyysisesti, kytkimillä ja johdoilla, mutta laitelähtöinen ohjelmointi oli vaikeaa ja virheiden korjaaminen työlästä. Matemaatikko John von Neumann sai kunnian 1945 julkaistusta raportista, joka kuvasi ENIACin korvaavan tietokoneen suunnitelman. Suunnitelmassa määriteltiin tietokoneen arkkitehtuuri (von Neumann arkkitehtuuri), jossa suoritettava ohjelma oli samassa muistissa käsiteltävän tiedon kanssa. Arkkitehtuuri sisälsi myös lasku-, ohjaus-, syöte- ja tulosteyksiköt. Ohjelman komentojen ja tiedon esitystapa perustui binäärilukuihin. Eri yksiköitä yhdisti väylä, joka mahdollisti tiedonsiirron tietokoneen osien välillä (ks. Kuva 14). Ohjausyksikkö ohjaa mm. tiedonsiirtoa muistista laskuyksikölle ja laskuyksiköltä takaisin muistiin. Ohjelman syöttäminen muistiin, tietokoneen yksiköitä yhdistävä väylä ja binäärinen tiedon esitystapa yksinkertaisti tietokoneen rakennetta merkittävästi.



Kuva 14: von Neumann arkkitehtuuri (von Neumann 1945) on ensimmäinen tietokonetta kuvaava arkkitehtuuri, jossa data ja ohjelma ovat samassa muistissa. Von Neumann arkkitehtuuri koostuu lasku-, ohjaus-, muisti-, syöte- ja tulosteyksiköistä. Arkkitehtuurin eri yksikköjä yhdistää väylä, joka toimii sekä ohjauksen että tiedon välittäjänä. von Neumann arkkitehtuuri on eniten käytetty tietokonearkkitehtuuri. Nykyään laskuyksikkö ja ohjausyksikkö ovat osa tietokoneen keskusyksikköä.

Von Neumannin arkkitehtuuri on yhä meidän nykyisten tietokoneidemme käyttämä arkkitehtuuri. Samalla teknologiat, joilla arkkitehtuuri on toteutettu, ovat kasvaneet teholtaan ja pienentyneet kooltaan. Tietokoneen ohjelmointi tapahtui alun perin konekielellä eli määrittelemällä binääriluvut, jotka saivat tietokoneen tekemään halutun asian. Seuraava askel ohjelmoinnissa oli käyttää sanoja ja desimaalijärjestelmän numeroita edustamaan konekielen komentoja ja syötteitä. Kääntäminen konekielelle tehtiin käsin, mutta ohjelman kirjoittaminen nopeutui kuitenkin merkittävästi. Sitten keksittiin, että sanat ja numerot voitiin antaa syötteenä tietokoneelle, jonka sisältämä ohjelma käänsi ne automaattisesti. Kääntäminen oli helppoa koska kyseessä oli vain ihmiselle yksinkertaisempi tapa kuvata konekieltä. Nykyajan ohjelmointikielet ovat huomattavasti korkeammalla abstraktiotasolla, joten yhtä komentoa vastaa usein joukko konekielisiä komentoja. Ohjelmointia on myös helpottanut syötteen ja tulosteen mahdollistavien laitteiden kehitys.

4.2 OHJELMOINNILLISET/ALGORITMISET PELIT

Pelit ovat luonteva ja motivoiva tapa harjoitella ohjelmoinnillista ajattelua. Suurin osa tarjolla olevista ohjelmoinnillista ajattelua tukevista peleistä keskittyy toimintaohjeisiin, joissa pitää muodostaa annetuista käskyistä tietyn ongelman ratkaiseva kokonaisuus. Ohjelmoinnillista ajattelua voi oppia myös erityyppisillä pulmapeleillä ja toisaalta myös ohjelmointityökaluja sisältävillä maailmanrakennuspeleillä.

4.2.1 TOIMINTAOHJEIDEN MUODOSTAMINEN PELINÄ

Monissa ohjelmoinnillista ajattelua käsittelevissä peleissä muodostetaan koodia tai toimintaohjeita. Pelissä on tyypillisesti hahmo, jota voi ohjata annetuilla käskyillä, esimerkiksi nuolilla. Hahmon pitää edetä pelialueella annettuun kohtaan. Käytettävien käskyjen määrä saattaa olla rajattu. Pelissä voi aueta uusia käskyjä sitä mukaa kun pelaaja selvittää tasoja. Käyttäjä muodostaa annetuilla käskyillä

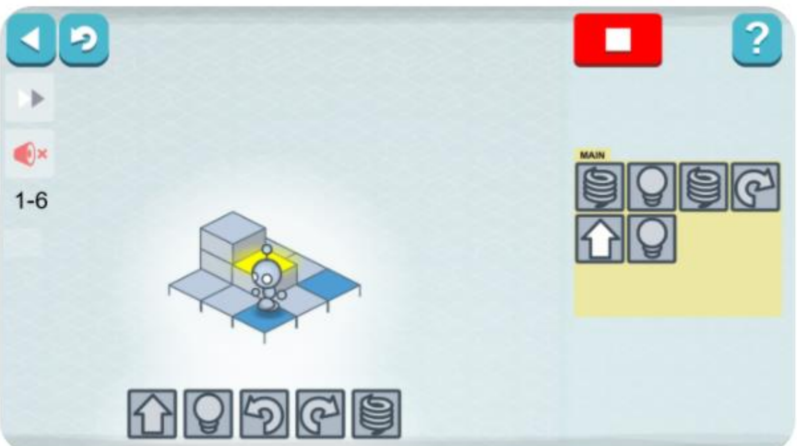
toimintaohjeita/ohjelmia annettujen ongelmien ratkaisemiseen, joten pienimuotoisesti kyseessä on ohjelmointi.

Jotkin pelit käyttävät ainoastaan liikkumisen ohjaamiseen liittyviä käskyjä. On kuitenkin hyvä, jos pelissä on mukana jonkinlainen toisto: esimerkiksi uudelleenkäytettävän aliohjelman välityksellä, näin harjoittelu ei rajoitu pelkkään peräkkäisyyteen ja pulmat voivat olla haastavampia.

Seuraavassa on esitelty muutama alakouluun sopiva peli:

LightBot (https://lightbot.com/) on tableteilla toimiva peli, jossa ohjataan valoja sytyttävää robottihahmoa (ks. Kuva 15). Pelissä on suomenkielisiä ohjeruutuja.

- ilmainen
- suomenkielinen
- iOS, Android



Kuva 15: LightBot

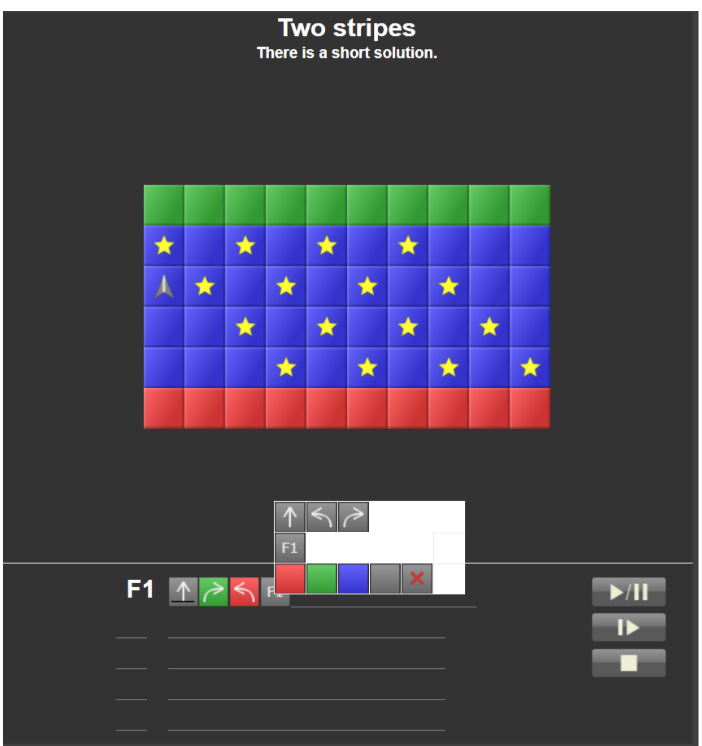
The Foos (http://thefoos.com/) on pienimmille oppilaille sopiva tablettipeli, jossa on mukana toimintaohjeita ja silmukkarakenteita (ks. Kuva 16).

- ilmainen
- kieliriippumaton
- iOS, Android

Kuva 16: The Foos

RoboZZle (http://www.robozzle.com/) on selainpohjainen ja tarjoaa aikuisellekin sopivia algoritmisen ajattelun pulmia (ks. Kuva 17). Soveltuu isommille oppilaille ja eriyttämiseen.

- ilmainen
- vaatii hieman englannintaitoa
- toimii selaimessa



Kuva 17: RoboZZle

ViLLE World on ViLLE-oppimisympäristön minipeli, jossa ohjataan autoa nuolilla (ks. Kuva 18). Käytössä on myös suorituksen ohjaaminen väritettävillä ruuduilla sekä funktioiden, eli aliohjelmien määrittely. ViLLE World pelejä on ViLLEn matematiikan opintopolkujen osana.

- ilmainen ViLLE-käyttäjille
- suomenkielinen
- toimii selaimessa

Kuva 18: ViLLE World

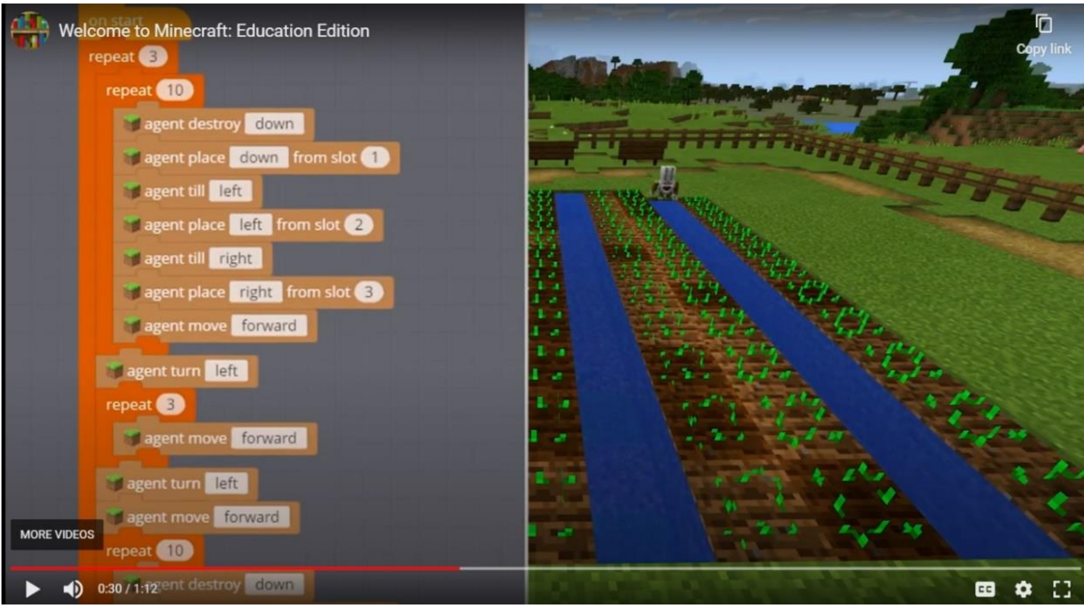
4.2.2 PELIN MUOKKAAMINEN TOIMINTOJA OHJELMOIMALLA

On pelejä, joissa pelaaja voi muokata peliympäristöä ja luoda pelisisältöä ohjelmoimalla. Käyttäjä voi itse ohjelmointityökalujen avulla muuttaa pelin logiikkaa ja lisätä toiminnallisuutta.

Minecraftissa on tarjolla Education Edition (ks. Kuvat 19 ja 20) ja siinä englanninkielinen Hour of Code, jossa käytetään ohjelmointilohkoja tai python koodia. Minecraftiä esittelevä Hour of Code esittelee hahmon, jota ohjataan Scratchista tutuilla lohkoilla. Minecraft Education editionia voi käyttää monipuolisesti oppimisalustana.

https://education.minecraft.net/lessons/coding-in-minecraft-an-intro

https://education.minecraft.net/blog/code-builder-command-blocks-and-more-come-to-education-edition

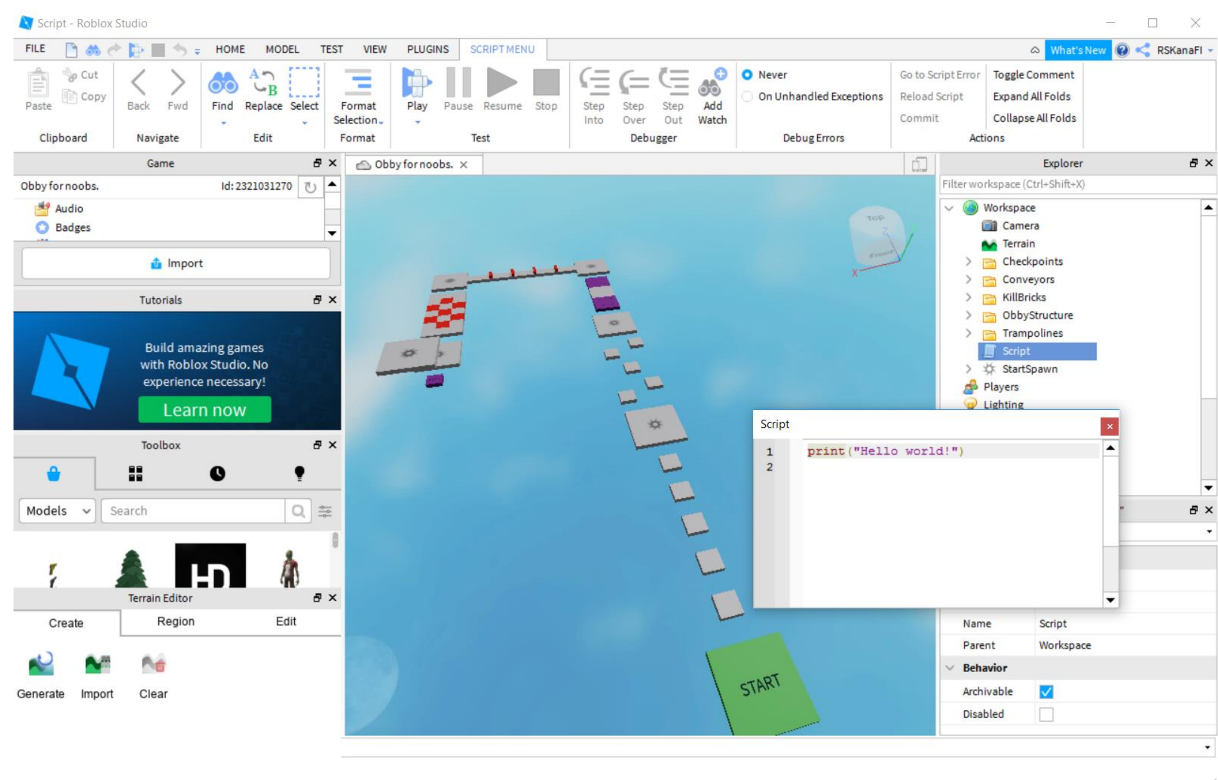


Kuva 19: MineCraft Education

Kuva 20: MineCraft Education

Roblox-pelissä käyttäjä voi luoda minipelejä (ks. Kuvat 21 ja 22), joita muut pelaajat voivat pelata. Suosituimmat pelit ovat syntyneet amatöörikoodaajien yhteistyönä.

Kuva 21: Roblox



Kuva 22: Roblox

4.3 GRAAFISET OHJELMOINTIYMPÄRISTÖT

Ohjelmointiympäristöissä on **editori**, johon ohjelma kirjoitetaan/rakennetaan ja **kääntäjä tai tulkki**, joka muuntaa ohjelmoijan kirjoittaman/rakentaman koodin konekielelle eli kielelle, jota tietokone "ymmärtää" ja kykenee suorittaa. Ohjelmointiympäristöjä on sekä graafisia että tekstipohjaisia.

Pääasiassa graafisia ympäristöjä käytetään ohjelmoinnin opetuksessa ja ohjelmistotuotannossa käytetään monimutkaisempia ja monipuolisempia tekstipohjaisia ympäristöjä. Englanniksi puhutaan IDE:istä (*Integrated Development Environment*). Ohjelmoijat saattavat käyttää esimerkiksi Eclipseä, IntelliJ:tä tai Visual Studiota. Peruskoulun opetuskäyttöön sopivat suppeammat (muutkin kuin graafiset) ohjelmointiympäristöt.

Opetuskäyttöön sopivia ohjelmointiympäristöjä määrittävät opetuskieli ja käytettävissä olevat resurssit. Suuri osa ympäristöistä (ja niiden materiaaleista) on kehitetty englanninkielellä, mutta niistä voi olla käännöksiä. Osa ohjelmointiympäristöistä toimii tableteille tai tietokoneille ladattavilla ohjelmilla, joitain käytetään selaimessa. Ohjelmointiympäristöjä on myös tehty robotin tai muun laitteen toiminnan ohjaamiseen.

Oppilaiden ohjelmoinnin oppimisen kannalta valitulla ohjelmointikielellä ei ole suurta merkitystä. Ohjelmoinnin perusteet ovat samat ohjelmointikielestä riippumatta ja tärkeintä onkin opettaa, miten perusrakenteet toimivat. Kun esimerkiksi silmukkarakenne on tuttu, on sitä helppo oppia käyttämään uudellakin ohjelmointikielellä tai uudessa ympäristössä, koska silmukan toimintaperiaate on sama. Ohjelmointiympäristö kannattaa valita siten, että opetuksen toteuttaminen oman ryhmän kanssa on helppoa ja sujuvaa, koska silloin oppilaatkin saavat opetuksesta suurimman hyödyn.

4.3.1 OHJELMOINTIYMPÄRISTÖJÄ

Alice (alice.org) on ilmainen tietokoneelle ladattava 3D-ohjelmointiympäristö, jossa koodi rakennetaan palikoilla. Ympäristö sopii esim. animaatioiden ja pelien luomiseen. Ympäristö on englanninkielinen ja siitä on kaksi versiota: Alice 2 sopii Alice-maailmaan tutustuville ja Alice 3 kokeneemmille ja vanhemmille oppilaille.

Micro:bit (microbit.org) on pieni korttitietokone, jossa on lamppuja ja painikkeita. Koodi kirjoitetaan toisella laitteella (tietokone, tabletti tai puhelin). Sitä voidaan ohjata eri ohjelmointikielillä, kuten Python ja Scratch. Micro:bittiä varten on myös kehitetty oma ohjelmointiympäristö MakeCode, jossa voi valita tekstipohjaisen ja graafisen ohjelmoinnin välillä.

Hopscotch (gethopscotch.com) on pelinkehitysohjelmisto 9–16 vuotiaille. Koodia rakennetaan graafisesti. Hopscotch ladataan Applen sovelluskaupasta ja sovellus on englanninkielinen.

Kodu (kodugamelab.com) on ilmainen pelienkehitysohjelma Windows-tietokoneille. Kodussa ohjelmoidaan graafisesti. Kodussa on myös valmiita materiaaleja eri ikäisille. Materiaalit ja ohjelmointiympäristö ovat saatavilla englanniksi.

Lego Mindstorms (lego.com/fi-fi/themes/mindstorms/about) on modulaarinen robotti, jonka toimintaa ohjataan erillisen sovelluksen avulla rakennetulla koodilla (perustuu Scratchiin). Robottiin saa ostettua erilaisia lisäosia, joten sillä voi tehdä vaikka mitä.

Pencil Code (pencilcode.net) on ilmainen selaimessa käytettävä ohjelmointiympäristö, jossa voi ohjelmoida sekä graafisesti että kirjoittamalla koodia (JavaScript, CSS, HTML). Ympäristö on englanninkielinen.

Scratch (scratch.mit.edu) on ilmainen selaimessa tai laitteelle ladattavalla ohjelmalla käytettävä ympäristö, jossa ohjelmoidaan graafisesti. Scratch sopii 8–16 vuotiaille ja se on saatavilla useilla kielillä (mm. suomeksi ja ruotsiksi).

Snap! (snap.berkeley.edu) on ilmainen Scratchistä tehty laajennos, joka sallii omien palikoiden rakentamisen ja tukee monipuolisemmin tietorakenteiden luomista. Snap! on ilmainen ja saatavilla suomeksi ja ruotsiksi niiltä osin, kun ne vastaavat Scracthiä, muutoin ympäristö on englanninkielinen.

Sphero (edu.sphero.com) on robotti, jota ohjataan piirtämällä, graafisesti tai kirjoittamalla JavaScriptiä. Koodia voi rakentaa sekä eri laitteille ladattavalla sovelluksella että selaimessa. Sphero on osittain suomennettu, mutta muutoin englanninkielinen. Tarjolla on erilaisia projekteja eri aiheista ja eri ikäisille.

Tynker (tynker.com) on ohjelmoinnin oppimisen ympäristö, jonka ohjelmointikieli perustuu Scratchiin. Tynkerissä on monenlaisia kursseja, jotka harjoittavat ohjelmoinnillista ajattelua ja koodaustaitoja. Tehtävää on eri ikäisille aina viisivuotiaista ylöspäin. Tynkeriä voi kokeilla maksuttomasti. Tynker on englanninkielinen.

4.4 TUTUSTUTAAN OHJELMOINNIN KONSEPTEIHIN

Ohjelmoinnissa on tiettyjä käsitteitä ja rakenteita (konsepteja), jotka toistuvat ohjelmointikielestä riippumatta. Näin ollen uusien ohjelmointikielten oppiminen on helppoa, jos konseptit ovat tuttuja. Konseptit ovat samoja niin graafisissa ohjelmointiympäristöissä kuin tekstipohjaisissa ohjelmointikielissä, joten oppilaittenkin kanssa kannattaa jonkin verran keskittyä näihin, jotta yläkoulussa tekstipohjaiseen ohjelmointiin on helpompi siirtyä.

Brennan ja Resnick (2012) ovat määritelleet, että ohjelmoinnin konsepteihin kuuluvat operaattorit, peräkkäisyys, samanaikaisuus, data, tapahtumat, ehtolauseet ja silmukat. Näiden lisäksi tärkeitä käsitteitä ovat lausekkeet ja lauseet, tyypit, muuttujat ja tietorakenteet.

(Tietokone)ohjelma koostuu (ohjelma)koodista, joka ei vielä yksinään tee mitään. Koodi tai ohjelma täytyy **ajaa** (englanniksi *run*), jotta tietokone **suorittaa** koodin. Koodia ei siis suoriteta samalla kun sitä kirjoitetaan vaan se suoritetaan vasta sitten, kun ohjelmoija päättää niin.

4.4.1 OPERAATTORIT, OPERANDIT, LAUSEKKEET JA LAUSEET

Ohjelmoinnissa **operaattoreiksi** kutsutaan merkkejä, joiden avulla luvuille, teksteille ja muille arvoille voidaan tehdä jotakin. Matemaattisia operaattoreita ovat mm. yhteen-, vähennys-, kerto- ja jakolaskut.

Niitä merkitään yleensä merkeillä +, -, * ja /, mutta merkintätapa voi vaihdella ohjelmointikielestä riippuen.

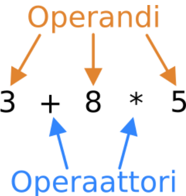
Tekstiä nimitetään ohjelmoinnissa merkkijonoiksi. **Merkkijonoissa** on nolla, yksi tai useampi merkki. Useimmiten merkkijonoilla on ohjelmointikielissä omat operaationsa. Esimerkiksi joissain kielissä matemaattinen operaattori + voi yhdistää kaksi merkkijonoa toisiinsa.

Ohjelmoinnissa käytetään usein myös vertailuoperaattoreita, jotka ovat myös matematiikasta tuttuja: suurempi kuin, pienempi kuin ja yhtä suuri kuin. Tärkeitä ovat myös loogiset operaattorit “ja”, “tai” ja “ei” (ks. Taulukko 4).

Taulukko 4: Loogisten operaattoreiden ja, tai ja ei totuustaulu. Lauseke “A ja B” on totta vain silloin, kun molemmat ovat totta ja muulloin epätotta. Lauseke “A tai B” on totta, jos toinen tai molemmat ovat totta ja muulloin epätotta. Lauseke “ei A” kääntää totuuden päinvastaiseksi.

A	**B**	**A ja B**	**A tai B**	**ei A**
tosi	tosi	tosi	tosi	epätosi
tosi	epätosi	epätosi	tosi	epätosi
epätosi	tosi	epätosi	tosi	tosi
epätosi	epätosi	epätosi	epätosi	tosi

Operaattorit eivät kuitenkaan yksinään riitä: pelkkä +-merkki ei vielä anna tulosta. Niiden ympärille tarvitaan **operandeja** eli arvoja, joille operaatio suoritetaan. Operaattorit ja operandit muodostavat yhdessä **lausekkeita** (ks. Kuva 23).



Kuva 23: Lausekkeet koostuvat operaattoreista ja operandeista.

Ohjelmoinnissa lausekkeillakaan ei vielä oikeastaan tee paljoakaan. Ohjelmoinnissahan tarkoitus on ohjata tietokonetta, joten tarvitaan käskyjä, joita tietokone ymmärtää. Käskyjä nimitetään **lauseiksi**. Esimerkiksi jos ohjelmointikielessä on määritelty lause “tulosta ruudulle”, voisi koodiin kirjoittaa “tulosta ruudulle 5 + 8”, jolloin ruudulle tulostuisi 13, kun koodi suoritetaan. Lauseke siis lasketaan ensin ja sitten lause suoritetaan lasketulla arvolla.

4.4.2 PERÄKKÄISYYS JA SAMANAIKAISUUS

Peräkkäisyys tarkoittaa sitä, että lauseet suoritetaan peräkkäin. Lauseet siis suoritetaan siinä järjestyksessä, kun ne on kirjoitettu ohjelmakoodiin.

Samanaikaisuus tarkoittaa sitä, että kahta tai useampaa eri koodia suoritetaan samaan aikaan.

4.4.3 TAPAHTUMAT

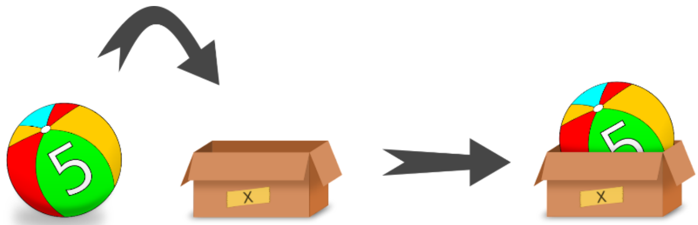
Ohjelmakoodin suoritus alkaa, kun ohjelma käynnistetään. Jotkin koodin osat voivat kuitenkin vaatia vielä jonkin **tapahtuman**, jotta ne suoritetaan. Esimerkiksi nettisivuilla olevan painikkeen klikkaaminen on tapahtuma, jonka jälkeen jonkin koodin suoritus alkaa. Ohjelma voi olla siis käynnissä ja odottaa, että jokin tapahtuma määrittelisi, mitä sen tulisi tehdä.

4.4.4 DATA, TYYPIT, MUUTTUJAT JA TIETORAKENTEET

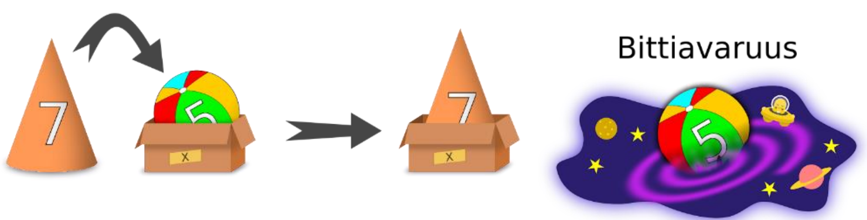
Tietokoneiden hyödyllisyys perustuu niiden nopeuteen käsitellä suuria määriä tietoa. **Tieto** eli **data** ja sen hallinta onkin ohjelmoinnissa erityisen tärkeää.

Ohjelmoinnissa tieto on aina jonkin tyyppistä. Tieto**tyyppejä** ovat esimerkiksi luvut (usein vielä kokonaisluvut ja liukuluvut (liukuluvuilla viitataan desimaalilukujen esitystapaan tietokoneessa) erikseen), merkkijonot ja totuusarvot. Ohjelmointikielestä riippuen tietotyypit esitetään joko eksplisiittisesti tai sitten tyypitystä ei näe ohjelmakoodista. Operaatioiden toimimiseksi on ohjelmoijan kuitenkin aina tiedettävä, minkä tyyppisillä arvoilla operaatiota suoritetaan. Joissain tapauksissa väärän tyyppinen arvo lausekkeessa voi keskeyttää ohjelman suorituksen tai estää suorituksen aloittamisen kokonaan.

Ohjelmoinnissa tieto tallennetaan muuttujiin. Ohjelmoinnin muuttujat eroavat matematiikasta tutuista muuttujista täysin. Ohjelmoinnissa **muuttuja** on ikään kuin tiedon nimi tai säilytyslaatikko. Muuttujiin voidaan tallentaa arvoja (eli esimerkiksi lukuja ja merkkijonoja, ks. Kuva 24) ja myöhemmin arvo voidaan hakea muistista käyttöön muuttujan nimellä. Muuttujien arvoja voidaan myös muuttaa eli vaihtaa ohjelman suorituksen aikana (ks. Kuva 25). Esimerkiksi pelissä voidaan pitää kirjaa saaduista pisteistä muuttujassa ja pelaajan saatua pisteen muuttujan arvoa korvataan arvolla, joka on yhtä pistettä suurempi.

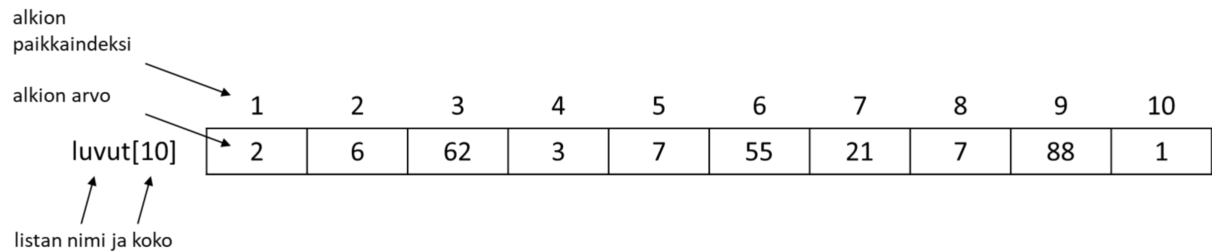


Kuva 24: Muuttujat voidaan ajatella säilytyslaatikoina, joihin voidaan tallentaa arvoja. Kuvassa x on muuttujan nimi ja 5 muuttujan arvo. Kun arvo on tallennettu muuttujaan, voidaan muuttujan nimellä palauttaa arvo käyttöön myöhemmin koodissa.



Kuva 25: Muuttujien arvoja muutettaessa edellinen arvo korvataan uudella, jolloin edellinen arvo ei ole enää tallennettuna minnekään, eikä siten käytettävissä. Muutoksen jälkeen koodissa päästään muuttujan nimellä enää vain käsiksi uuteen arvoon 7.

Muuttujiin voi tallentaa ainoastaan yhden arvon kerrallaan, joten ohjelmoinnissa käytetään myös monipuolisempia tietotyyppejä nk. tietorakenteita. Ohjelmoinnin **tietorakenteita** ovat mm. taulukot, listat ja puut. Näistä tärkein on **lista**, joka on dynaaminen rakenne, johon **alkiot** eli tietoyksiköt tallennetaan johonkin tiettyyn järjestykseen (ks. Kuva 26). Dynaamisuudella tarkoitetaan sitä, että listan järjestystä voidaan muuttaa, siihen voidaan lisätä alkioita, siitä voidaan poistaa alkioita ja alkioiden arvoja voidaan muuttaa. Listan alkioihin voidaan viitata järjestysnumerolla eli **indeksillä**. Scratch-ohjelmoinnissa lista alkaa ykkösestä eli ensimmäinen alkio löytyy indeksillä 1. Lista sisältää yleensä määritellyn kokonaisuuden, esimerkiksi kuukaudet, viikon sademäärät tai seitsemän kertotaulun tulokset. Listan alkioiden järjestys voi olla kasvava, laskeva tai esimerkiksi alkion lisäysjärjestyksen mukainen.

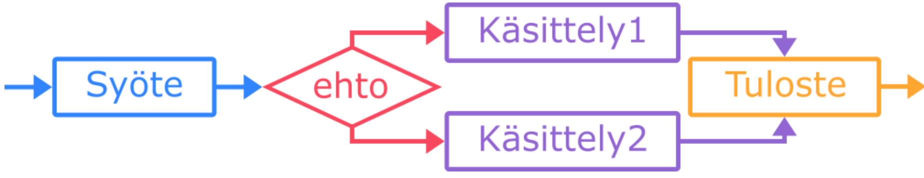


Kuva 26: Lista on dynaaminen tietorakenne, jonka alkioita voidaan lisätä ja poistaa. Se on yksi ratkaisu useamman yhteenkuuluvan tiedon tallentamiseen. Listan kullakin alkiolla on paikkaindeksi, jonka avulla voidaan suoraan päästä kyseiseen tietoon. Scratch-ohjelmointikielessä lista alkaa indeksistä yksi, mutta monessa ohjelmointikielessä se alkaa nollasta. Listan alkiot ovat itsenäisiä, joten niiden arvo ei ole riippuvainen toisten alkioiden arvoista.

4.4.5 EHTOLAUSEET

Ohjelman suoritus ei ole aina lineaarista (ks. Kuva 27), vaan lauseiden suoritus voi myös olla ehdollista. **Ehtolauseessa** tarkistetaan, pitääkö jokin ehto paikkansa ja sen perusteella joko suoritetaan tai jätetään suorittamatta jokin koodipätkä. Usein ehdon jälkeen laitetaan myös vaihtoehtoinen koodipätkä. Tällöin ensimmäinen koodipätkä suoritetaan, jos ehto on totta, mutta jos se ei ole totta, suoritetaan vaihtoehtoinen koodipätkä (ks. Kuva 28).

Kuva 27: Ohjelman lineaarisessa suorituksessa jostakin tilanteesta, syötteestä, päästään lopputulokseen, tulosteeseen, aina samanlaisen käsittelyn kautta.



Kuva 28: Ehtolause määrittelee vaihtoehtoisen suoritusketjun. Jos ehto on totta, suoritetaan käsittely 1 ja jos ehto ei ole totta, suoritetaan käsittely 2.

Käytännössä ehtolauseet ovat "JOS-lauseita": JOS ehto on tosi, NIIN suoritetaan koodipätkä A, MUUTOIN suoritetaan koodipätkä B. Ehtolauseissa valinta tehdään vertailu- ja loogisilla operaattoreilla, joiden lopputuloksena on arvo tosi tai epätosi.

Esimerkki:

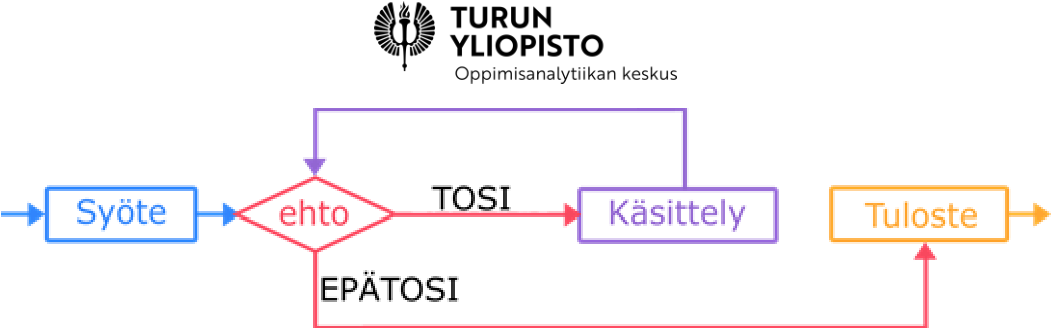
```
Luku A on 5.
Luku B on 8.
JOS luku A ON SUUREMPI KUIN luku B,
NIIN vähennä luvusta A yksi,
MUUTOIN vähennä luvusta B yksi.
```

Suorituksen jälkeen luku A on edelleen 5, mutta luku 7, koska ehtolauseesta suoritettiin vaihtoehtoinen koodipätkä, jossa luvusta B vähennetään yksi.

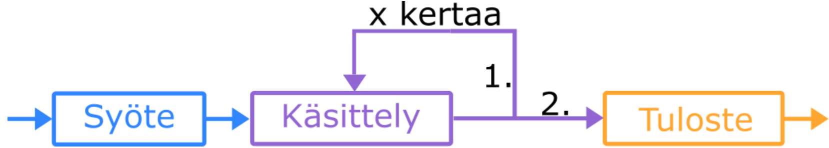
4.4.6 SILMUKAT

Joskus ohjelmoidessa törmää tilanteeseen, jossa samaa koodipätkää kirjoitetaan monta kertaa peräkkäin. Tällöin silmukkarakenteesta tulee hyödyllinen. **Silmukoiden** avulla tiettyä koodipätkää voidaan suorittaa monta kertaa kirjoittamatta lauseita yhä uudelleen. Ohjelman suoritus siis toistaa tiettyä koodipätkää monta kertaa, minkä takia silmukoita kutsutaan myös toistolauseiksi.

Mistä sitten tiedetään, kuinka monta kertaa silmukan koodipätkää suoritetaan? Silmukoita on pääasiassa kahta tyyppiä: ehdollista ja ehdotonta. Silmukan suoritus voidaan sitoa johonkin ehtoon: silmukan suoritusta jatketaan niin kauan, kunnes sitä ohjaava ehto muuttuu epätodeksi (ks. Kuva 29). Tai toisaalta silmukan suoritukselle voidaan antaa lukumäärä: silmukka suoritetaan tasan näin monta kertaa (ks. Kuva 30).



Kuva 29: Ehdollista silmukkaa suoritetaan niin kauan, kun määritelty ehto on voimassa. Jotain on siis muututtava silmukan sisällä, jotta ehto muuttuu epätodeksi ja silmukka päättyy.



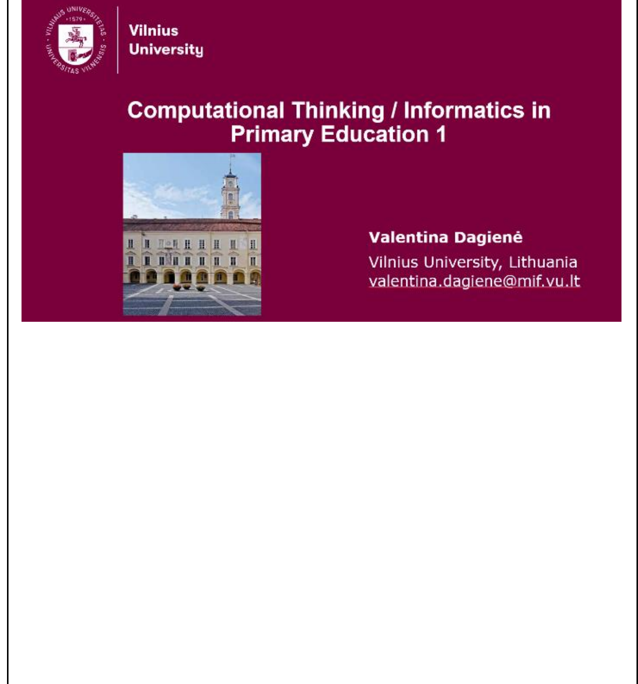
Kuva 30: Ehdotonta silmukkaa suoritetaan aina tietyn lukumäärän verran. Kuvassa siis käännytään ensimmäiseen haaraan niin monta kertaa kuin koodissa on määritelty (x).

5 TYÖPAJA 2: TUTUSTUTAAN GRAAFISEEN OHJELMOINTIIN

Työpajan tavoitteena oli esitellä, miten luodaan ohjelmoinnillista ajattelua ilman tietokonetta (nk. unplugged) harjoittavia tehtäviä, ja tutustuttaa ohjelmointiin Scratch-ohjelmointiympäristössä. Ohjelmoinnillisen ajattelun harjoitusten luominen perustui professori Valentina Dagienen luentoon. Häntä voidaan pitää unplugged-harjoitusten kehittämisen ammattilaisena, koska hänen luoma kansainvälinen Bebras-kilpailu perustuu näihin ja lisäksi hän on suunnitellut unplugged-harjoituksia useaan oppikirjaan Liettuassa. Työpajassa siirryttiin myös alkuopetuksen unplugged-harjoituksista graafiseen ohjelmointiin. Ohjelmointiympäristönä käytettiin Scratch-ohjelmointikielen mukana tulevaa ympäristöä. Perusohjelmoinnin käsitteiden osaamisen lisäksi on tärkeää oppia Scratch-ympäristön tarjoamat erilaiset mahdollisuudet, koska silloin voidaan helposti luoda näyttäviä ja hauskoja ohjelmia.

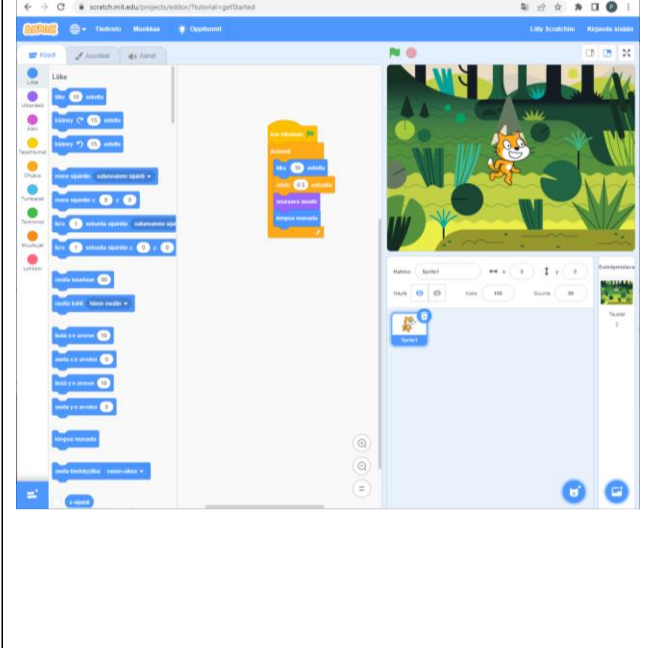
Työpaja alkoi lyhyellä alustuksella, jossa esitettiin yhteenveto edeltävästä verkkosisällöstä ja käytiin lävitse edellisen työpajan kotitehtävien vastaukset. Ensimmäisessä osiossa käsiteltiin ohjelmoinnillista ajattelua ja millaisia tehtäviä voidaan luoda sen harjoittamiseen. Toisessa tutustuttiin Scratch-ohjelmointiympäristön ominaisuuksiin ja sen käyttämiseen ohjelmoinnissa. Kolmannessa osiossa tehtiin yksinkertaisia ohjelmia.

Valentina Dagiene ohjelmoinnillinen ajattelu ja unplugged-harjoitusten luominen

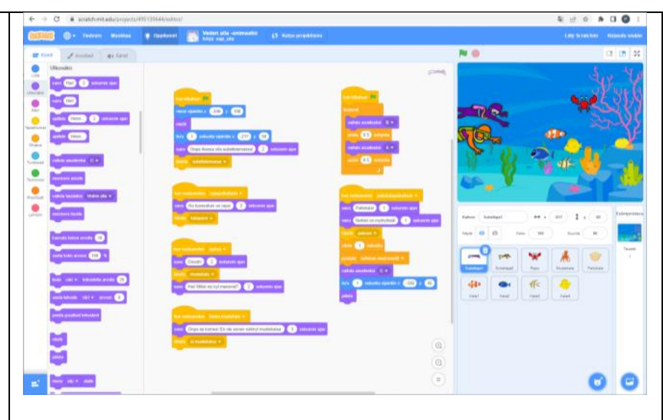
• Mitä on ohjelmoinnillinen ajattelu? • Ohjelmoinnillisen ajattelun taidot • Ohjelmoinnillisen ajattelun käsitteet • Ohjelmoinnillisen ajattelun opettaminen • Bebras (Majava)-kilpailu • Ohjelmoinnillisen ajattelun opettaminen Liettuassa	

• Millaisia unplugged tehtäviä voidaan luoda harjoittamaan ohjelmoinnillista ajattelua?	

Scratch-ohjelmointiympäristö

• Ohjelmointiympäristön kieli • Koodi -välilehti o Lohkoryhmät o Lohkot o Työskentelyalue o Näyttämö o Hahmot o Esiintymislava • Asusteet/Taustat -välilehti • Äänet • Hahmo -kirjasto • Tausta -kirjasto • Lisää laajennus	

Scratch-ohjelmointia

• Ohjelmoiminen Scratchillä • Näyttämön koordinaatisto • Hahmojen liikkuminen • Hahmojen ulkonäkö • Äänet	

• Tapahtumat	

6 VERKKOSISÄLTÖ 3: SCRATCH-OHJELMOINTI JA SEN SOVELTAMINEN MATEMATIIKASSA

Tässä osiossa käsitellään ohjelmointia ja sen soveltamista matematiikan oppimiseen. Osio alkaa tietokoneen ominaisuuksien hyötyjen läpikäynnillä matematiikan oppimisen kannalta. Ohjelmoinnista kerrataan ensin Scratch-ympäristön piirteet ja sitten jatketaan työpajassa aloitettua ohjelmointia käymällä lävitse aikaisemmin esiteltyjen keskeisten ohjelmoinnin konseptien toteutus Scratchissä. Lisäksi esitellään kaksi uutta konseptia. Lopuksi tarkastellaan Scratch-ympäristön hyödyntämisen mahdollisuuksia matematiikan oppitunnilla.

6.1 TIETOKONEEN HYÖTY MATEMATIIKALLE

Miksi matematiikan opiskelussa on hyötyä tietokoneen ja ohjelmoinnin käyttämisestä?

Tietokone ❤ matematiikka

Useimmat arkielämän matematiikan käyttökohteet ovat nykyään älylaitteella tai tietokoneella. Jokainen käyttää verkkopankkia ja useimmat tekevät ostoksia netissä. Matematiikkaa työssään tarvitsevat ammattilaiset, esimerkiksi insinöörit, kirjanpitäjät, matemaatikot tai luonnontieteilijät, käyttävät tietokonesovelluksia. Monet hyödyntävät taulukkolaskenta- ja tietokonealgebraohjelmia, jotka ovat tuttuja jo lukiolaisille, mutta ammattilaiset käyttävät myös erikoissovelluksia esimerkiksi suunnitteluun ja simulointiin.

Matematiikka on kieli, jolla voimme mallintaa fyysisen maailman toimintaa. Kaikki teknologiset saavutuksemme on tehty matematiikan avulla. Ennen nykyaikaa ihminen laski ja suunnitteli kynällä ja paperilla, mutta nykyään apuna on tietokoneiden laskentateho. Pystymme laatimaan tarkkoja ja laajoja malleja, joiden avulla saadaan lisää tietoa todellisuudesta ja suunnitellaan teknisiä uudistuksia.

Esimerkkejä tietokoneiden mahdollistamista teknisistä ja tieteellisistä saavutuksista:

- ihmisen genomin sekvensointi, uusien lääkkeiden kehittäminen sen avulla
- paikannus satelliittien avulla
- vähäpäästöisen auton moottorin polttoprosessia ohjaava sulautettu järjestelmä
- internet ja mobiilitiedonsiirto
- teollisuusprosessien optimointi raaka-aineiden hyödyntämiseksi tehokkaammin

Käytämme myös arjessa jatkuvasti tietokoneita ja niillä tehtyjä matemaattisia sovelluksia, esimerkkeinä vaikka auton peruutustutka ja pesukoneen älykäs pesuohjelma. Hyödyn lisäksi on saatu myös huvia. Digitaalisesti tuotettuja elokuvia voi katsoa suoratoistopalvelussa ja suositella sitten somessa kaverillekin.

Matematiikka ja tietokoneet ovat aikamoinen menestystarina!

6.1.1 TIETOKONE MATEMATIIKAN OPETUKSEN APUVÄLINEENÄ

Alakoululaiselle tärkein matematiikan työväline on yhä kynä ja paperi. Tietokone tarjoaa kuitenkin matematiikan opetukseen isoja etuja. Matematiikan ylioppilaskirjoitukset on toteutettu sähköisesti jo muutaman vuoden ajan ja koejärjestelmässä käytössä olevia ohjelmistoja, esimerkiksi dynaamista geometriaa ja symbolista laskentaa yhdistävää Geogebra-ohjelmistoa ja symbolisen laskennan ohjelmistoa Maximaa, hyödynnetään koko lukion ajan. Myös muilla kouluasteilla käytetään yhä enemmän interaktiivisia ohjelmistoja.

Ohjelmistojen käyttö matematiikan opetuksessa on monin tavoin perusteltua. Sitä puoltaa helppokäyttöisyys, oppijan motivointi ja erityisesti mahdollisuus tukea matematiikan usein abstraktien käsitteiden ymmärtämistä. On paljon tutkimuksia, joissa on todettu, että opetusteknologian käytöllä saavutetaan parempia oppimistuloksia. Myönteiset tulokset on saatu tutkimalla eri matematiikan osa-alueita (geometria, murtoluvut…) ja eri menetelmiä (todistaminen, päättely…). Näiden lisäksi teknologian käyttö on vaikuttanut positiivisesti opiskelijoiden motivaatioon ja asenteeseen matematiikan oppimista kohtaan. Laajuudesta ja tutkittavasta osa-alueesta riippumatta on havaittu **visualisoinnin** olevan keskeistä, jotta oppilas ymmärtää paremmin opetettavia käsitteitä.

Visualisointi ei ole pelkästään jonkun toisen luoma kuva tilanteesta, vaan oppijan itsensä koostama malli, jonka avulla hän voi tarkastella eri seikkojen vaikutusta ongelmaan. Visualisoinnin avulla tuodaan näkyväksi muuten piiloon jääviä piirteitä. Tavoitteena on, että samalla kun oppilas luo visualisoinnin hän joutuu pohtimaan matematiikan osa-aluetta tai käsillä olevaa ongelmaa, jotta saisi sen oikein esitettyä. Esitys puolestaan antaa kokonaiskuvan ja nostaa esille tärkeät asiat.

Tietokonetta voidaan käyttää samaan kuin kynää ja paperiakin, tai voidaan hyödyntää sitä tuomaan opetukseen jotakin uutta ja rakentavaa. Uudenlaisessa lähestymistavassa on kaksi haastetta: opettajan ja oppijoiden täytyy hallita uuden välineen käyttö ja toisaalta täytyy huomata ero tehokkaan käytön ja oppimisen tarpeiden välillä. Oppilailla on taipumusta ajatella itsensä matematiikan käyttäjinä ja käyttää apuvälineitä oikoteinä, välttyäkseen oppimiseen liittyvältä kognitiiviselta rasitukselta.

Uudenlaiset digitaaliset työkalut voidaan jakaa kuuteen kategoriaan, jotka kaikki liittyvät opetuksen ja oppimisen siirtymiseen kohti itse tekemiseen perustuvaa oppimista.

- dynaaminen ja graafinen
- laskentatehoa hyödyntävä
- uudella tavalla matematiikan aiheita esittävä
- koulumatematiikan ja oppilaan kokemuspiirin yhdistävä
- yhteisen tiedonluomisen mahdollistava
- opettajan uudenlaista roolia tukeva

Dynaamisilla graafisilla työkaluilla voi tutkia matemaattisia käsitteitä. Ohjelmat mahdollistavat helposti arvojen ja erilaisten laskutapojen kokeilemisen, jolloin voidaan havainnoida kaavojen käyttäytymistä. Visualisointi erilaisilla kuvaajilla auttaa tulosten hahmottamisessa.

Laskentatehoa hyödyntävät työkalut tuovat uuden näkökulman. Tietokoneella voidaan automatisoida monimutkaiset laskutoimitukset tai suurten datamäärien käsittely. Huomio siirtyykin laskemisesta laskemisen määrittelyyn.

Työvälineet, jotka tarjoavat **uudenlaisia tapoja esittää matematiikan aiheita** muuttavat käsitystä siitä, mitä voimme oppia. Sujuva geometriaohjelmiston käyttö saattaa helpottaa geometrista ajattelua myös silloin kun ohjelmisto ei ole käytössä ja ennen päättelyä vaatineet käsitteet saattavat hahmottua oppijan mielessä selkeinä ilman formaaleja todistuksiakin.

Digitaalisilla työkaluilla voidaan **luoda siltaa koulumatematiikan ja oppilaan kokemuspiirin** välillä. Näin voidaan luoda mukaansatempaavia ympäristöjä, joissa oppija houkutellaan ajattelemaan matemaattisesti. Tutkimuksissa on käytetty esimerkiksi pelinrakentamista osana matematiikan opetusta.

Yhteistyö netissä, yhteinen tiedonluominen, joko oman luokan sisällä tai koulujen välillä, laajentaa luokkahuoneen nettiin. Kun oppimisympäristönä on jaettu mikromaailma, muuttuu opettajan rooli perinteisestä kohti oppimista tukevan mahdollistajan ja mentorin roolia.

Oppimisjärjestelmät voivat tukea opettajaa ottamaan uuden roolin oppijan oman oppimisen **tukijana**, kun oppijat tutkivat uusia käsitteitä digitaalisten työvälineiden avulla.

Työvälineet voivat kuulua samanaikaisesti useaan kategoriaan. Luokittelusta on apua, kun tarkastellaan työvälineiden ominaisuuksia pedagogiselta kannalta. Ohjelmoinnissa voidaan toteuttaa matematiikan digitaalisten työkalujen ideoita. Siinä voidaan valita sellaisia ominaisuuksia, joissa toteutus muodostaa osan oppimisesta. Näin voidaan nähdä automaation takana olevat matematiikan ja ohjelmoinnin periaatteet.

Lähteet:

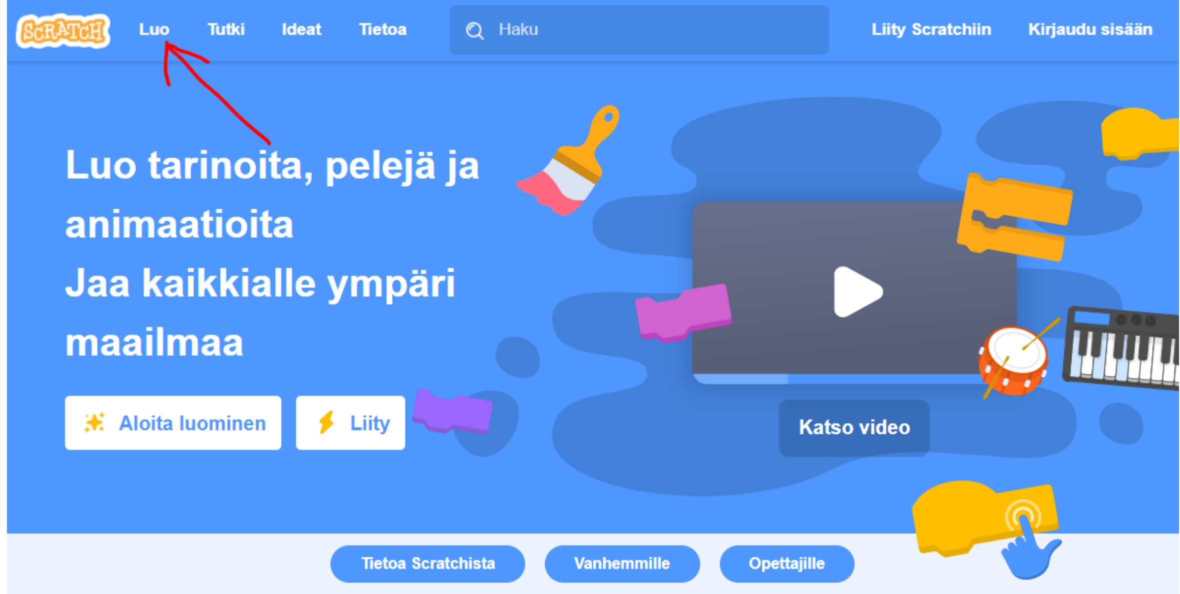
Celia Hoyles (2018) Transforming the mathematical practices of learners and Transforming the mathematical practices of learners and teachers through digital technology, DOI: 10.1080/14794802.2018.1484799

Monika Dockendorf (2020) How Can Digital Technology Enhance Mathematics Teaching and Learning?

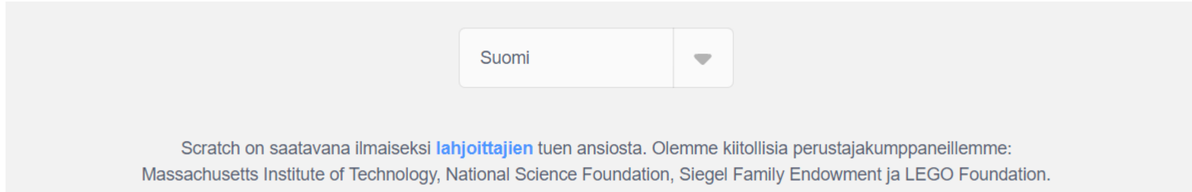
Examining Multiple Intelligences and Digital Technologies for Enhanced Learning Opportunities 2020 | book-chapter, DOI: 10.4018/978-1-7998-0249-5.ch01

6.2 SCRATCH-YMPÄRISTÖN ESITTELY

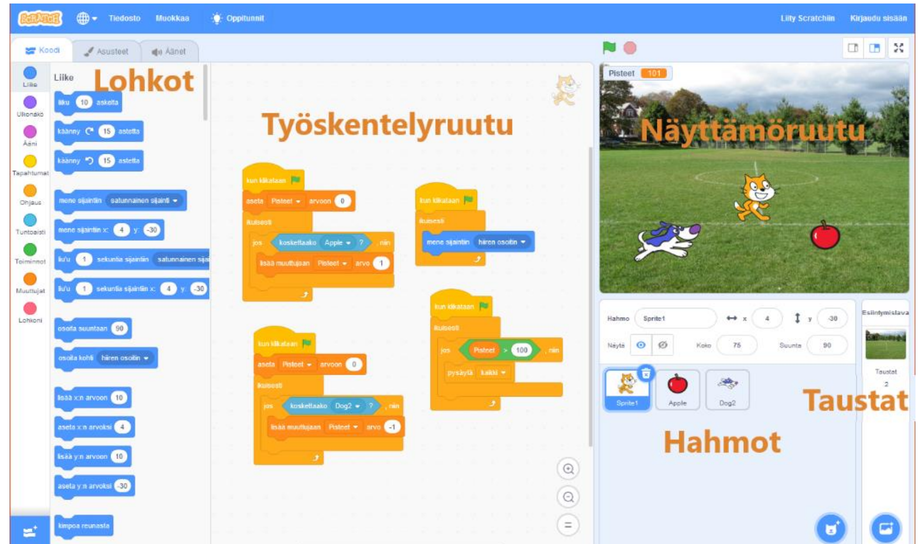
Scratch löytyy osoitteesta https://scratch.mit.edu/ ⟶ **Luo**.



Scratch-sivuston etusivun alareunassa on kielivalintapainike.



Scratch-ympäristö koostuu näyttämöruudusta, hahmot ja taustat -ruudusta ja työskentelyruudusta (ks. Kuva 31).



Kuva 31: Scratch-ympäristö koostuu työskentelyruudusta, näyttämöruudusta, hahmolistauksesta ja taustalistauksesta. Kun työskentelyruudun koodi-välilehti on auki, näkyy myös lohkolistaus.

Työskentelyruudulla on kolme eri välilehteä: **koodi**, **asusteet/taustat** (riippuen siitä, onko hahmo vai tausta valittuna) ja **ääni**. Koodi-välilehdellä on lohkolistaus ja tilaa hahmon koodille. Asusteet-välilehdellä voi muokata valittuna olevan hahmon asusteita. Taustat-välilehdeltä voi muokata ohjelman taustoja. Äänet-välilehdeltä voi muokata valittuna olevan hahmon tai taustojen ääniä.

Ohjelmakoodi kootaan siirtämällä lohkot listasta koodialueelle. Koodia voi rakentaa mille tahansa hahmolle ja taustalle. Muista olla tarkka siitä, mille hahmolle tai taustalle rakennat koodia! Koodilohkoja ei voi laittaa vääriin paikkoihin niiden muodon ansiosta.

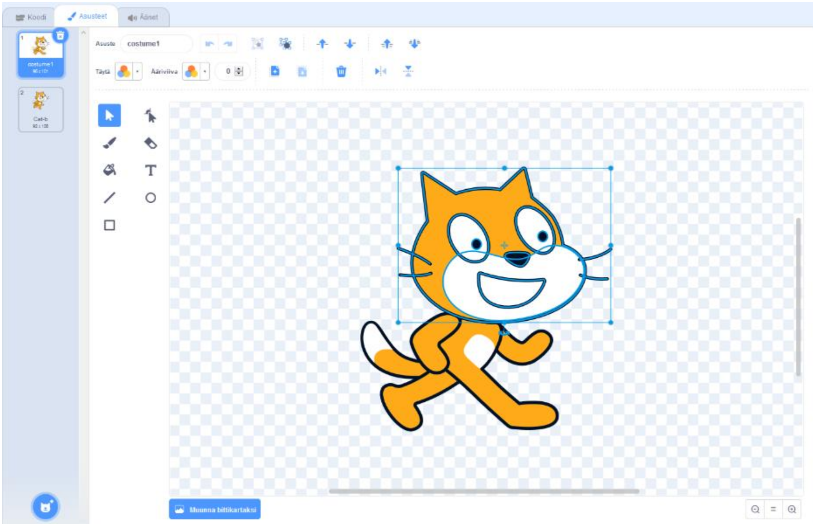
Näyttämö on koordinaatisto, jonka origo on näyttämön keskellä. Koordinaatisto on olemassa, vaikkei taustaksi valitsisikaan koordinaatistoa. Hahmot sijaitsevat jossakin pisteessä koordinaatistossa ja ne voivat liikkua näyttämöllä/koordinaatistossa eri tavoilla.

Hahmoja voi valita Scratchin omasta hahmokirjastosta, piirtää itse tai ladata tietokoneelta. Hahmot ovat toisistaan erillisiä elementtejä. Hahmojen ulkonäkö sisältää Scratchissa asusteiden lisäksi myös näkyviin tulevat ajatus- tai puhekuplat. Hahmon ei tarvitse olla sarjakuvahahmo, vaan se voi olla esimerkiksi piste, jonka liike piirtyy näyttämölle. **Taustoja** voi valita Scratchin omasta taustakirjastosta, piirtää itse tai ladata tietokoneelta. Voit valita taustaksi esimerkiksi jalkapallokentän tai koordinaatiston.

Hahmoilla ja taustoilla voi olla myös ääniä. **Ääniä** saa myös kirjastosta tai niitä voi nauhoittaa itse. Ääniä voi myös muokata Äänet-välilehdellä.

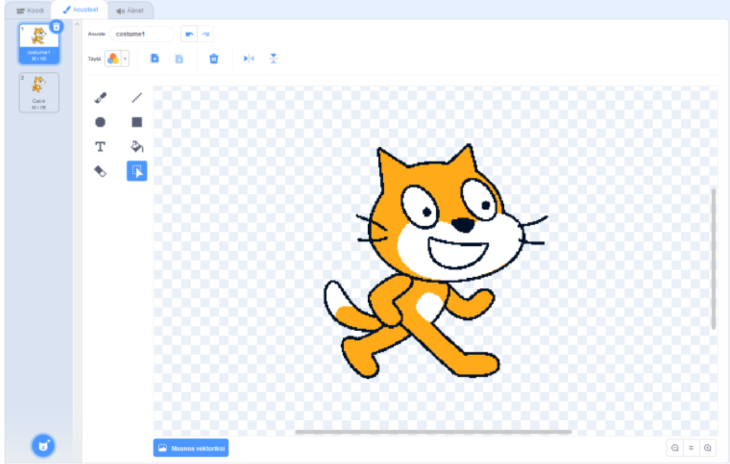
Tapahtumat aloittavat koodin suorituksen. Samalla hahmolla tai taustalla voi olla useita koodipätkiä, jotka reagoivat eri tapahtumiin.

Taustojen ja hahmojen piirtäminen ja muokkaaminen tapahtuu joko vektorikuva- tai bittikartta-muodossa. **Vektorikuvat** koostuvat muodoista, jotka voidaan esittää matemaattisina kaavoina (ks. Kuva 32). Vektorikuvat ovatkin helposti suurennettavissa ja pienennettävissä, koska kuvan kaavan mittasuhteet pysyvät samana, vaikka koko muuttuu.



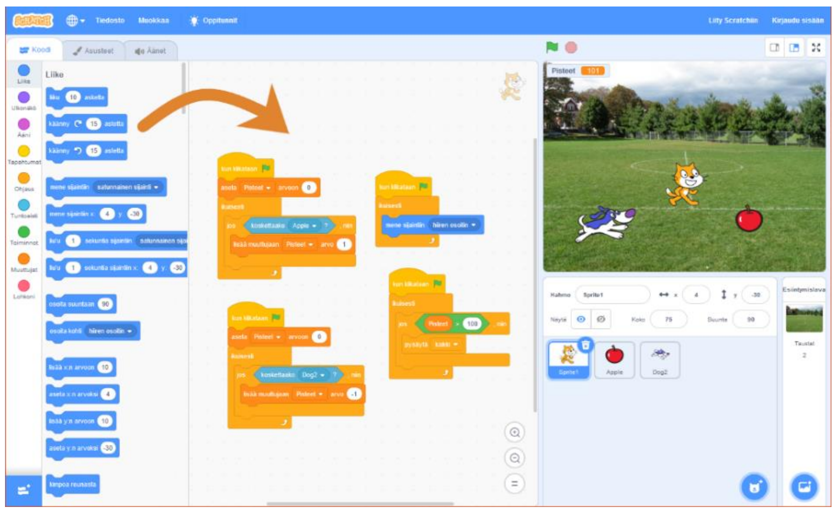
Kuva 32: Vektorikuvissa kuva koostuu elementeistä, jotka muodostetaan matemaattisten kaavojen avulla. Vektorikuvien elementtejä voi helposti muokata yksitellen (esim. väriä ja kokoa) ilman, että kuvan muut osat muuttuvat. Lopullisen kuvan kokoa voi myös muuttaa ilman, että kuvan laatu kärsii, koska kuva muodostetaan kaavojen avulla aina sopivan kokoisena.

Bittikarttakuvat taas koostuvat pikseleistä eli pienistä ruuduista, joille on annettu väri (ks. Kuva 33). Bittikarttakuvien suurentaminen ja pienentäminen useimmiten pilaa kuvan ja siksi bittikarttakuvat toimivatkin vain tietyn kokoisena. Vektorikuvan voi aina muuttaa helposti bittikarttakuvaksi, mutta bittikarttakuvaa ei voi muuttaa takaisin samanlaiseksi vektorikuvaksi. Esimerkiksi Scratchissä kissa koostuu oikeasti monista eri muodoista (vektorikuvana), mutta jos se muutetaan ensin bittikarttakuvaksi ja sitten takaisin vektorikuvaksi, siitä tulee yksi pikselinen muoto (voit vaikka kokeilla tätä itse!).



Kuva 33: Bittikarttakuvat koostuvat pikseleistä eli pienistä ruuduista, joilla on ominaisuutena vain väri. Bittikarttakuvat ovat aina tietyn kokoisia ja suurennettaessa niiden kuvanlaatu kärsii ja ne "pikselöityvät" eli pikselit erottuvat kuvassa neliöinä.

Scratchissa ohjelmat rakennetaan lohkoista, jotka ovat listattuna työskentelyruudun koodivälilehden vasemmassa reunassa. Ohjelmat kootaan raahaamalla halutut lohkot työskentelyruutuun (ks. Kuva 34). Ohjelmakoodit rakennetaan aina tietylle hahmolle tai taustalle eli eri hahmoilla ja taustoilla voi olla eri koodit. Yhdellä hahmolla tai taustalla voi myös olla useampi koodi (ks. Kuva 34).



Kuva 34: Scratchissä koodi rakennetaan raahaamalla lohkoja lohkolistasta työskentelyruudulle.

6.3 OHJELMOINNIN KONSEPTIT SCRATCHISSÄ

Aiemmissa osioissa käsiteltiin ohjelmoinnin konsepteja yleisesti. Konseptit olivat

- Operaattorit, operandit, lausekkeet ja lauseet,
- Peräkkäisyys ja samanaikaisuus,

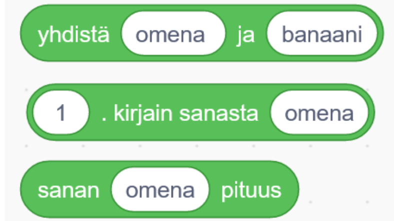
- Tapahtumat,
- Data, tyypit, muuttujat ja tietorakenteet,
- Ehtolauseet ja
- Silmukat.

Tässä käsitellään vielä kaikki konseptit Scratchillä toteutettuna.

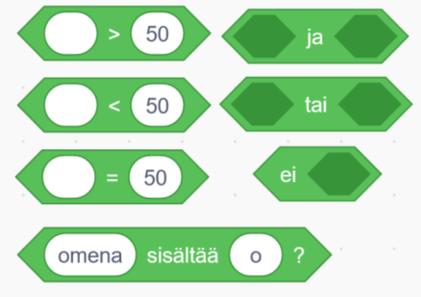
6.3.1 OPERAATTORIT, OPERANDIT, LAUSEKKEET JA LAUSEET

Scratchissä operaattorit on jo yhdistetty **lausekkeiksi** lohkokategoriaan Toiminnot. Pyöreäpäisten vihreiden lausekkeiden arvot ovat lukuja tai merkkijonoja ja teräväpäisten totuusarvoja eli tosi tai epätosi (ks. Kuvat 35–37). Lohkojen valkoisiin osuuksiin kirjoitetaan operandit.

Kuva 35: Scratchin matemaattiset lausekkeet

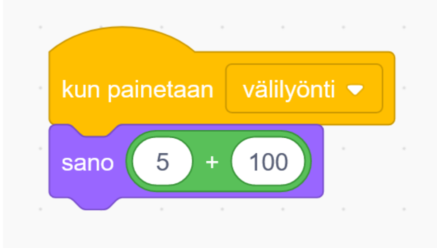


Kuva 36: Scratchin merkkijono-operaatiot



Kuva 37: Scratchin loogiset lausekkeet

Lauseita taas ovat ne lohkot, jotka sopivat tapahtumalohkon alle (ks. Kuva 38).



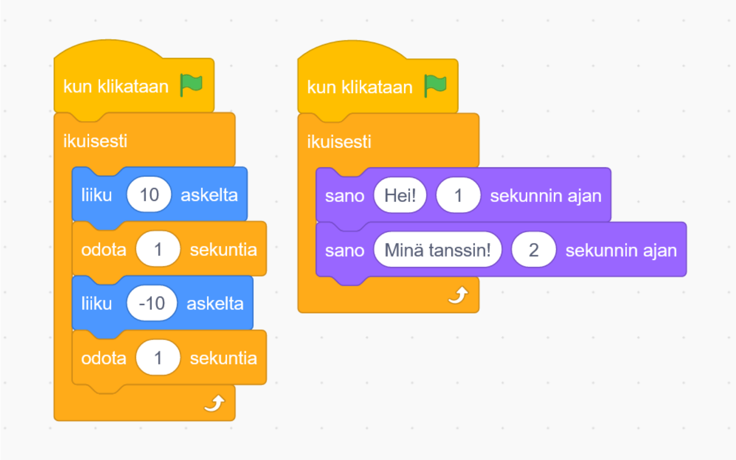
Kuva 38: Scratchissa lauseet ovat lohkoja, jotka sopivat toisten lohkojen alle .Niillä on yläreunassa kolo ja alareunassa uloke (esim. kuvassa keltaiseen tapahtumalohkoon on liitetty sano-lohko, joka sanoo sen sisältämän lausekkeen sisällön 5+100).

6.3.2 PERÄKKÄISYYS JA SAMANAIKAISUUS

Lohkot suoritetaan peräkkäin siinä järjestyksessä kuin ne ovat tapahtumalohkon alla (ks. Kuva 39).

Kuva 39: Lauseet eli lohkot suoritetaan peräkkäin eli lohko kerrallaan. Ensin hahmo liikkuu 10 askelta, sitten se odottaa 1 sekunnin tekemättä mitään, jonka jälkeen se sanoo "Hei!" puhekuplassa 1 sekunnin ajan. Sitten suoritus loppuu.

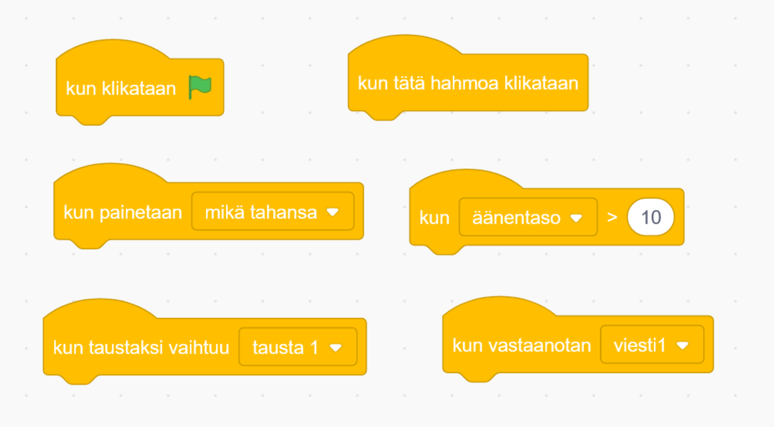
Sama tapahtuma voi aloittaa useamman koodin suorituksen (ks. Kuva 40). Koodit voivat olla samalla hahmolla, eri hahmoilla tai/ja näyttämöllä.



Kuva 40: Scratchissä useita koodipätkiä voidaan suorittaa samanaikaisesti, kun ne on liitetty samaan tapahtumaan. Tässä tapauksessa hahmo sekä liikkuu edes takaisin että sanoo vuorotellen "Hei!" ja "Minä tanssin!" kun Scratch-ympäristön vihreätä lippua klikataan.

6.3.3 TAPAHTUMAT

Scratchissä tapahtumia ovat lohkot, joiden yläpuolelle ei voi laittaa muita lohkoja (ks. Kuva 41). Niiden tarkoitus on käynnistää koodipätkän suoritus, kun jotain tiettyä tapahtuu.

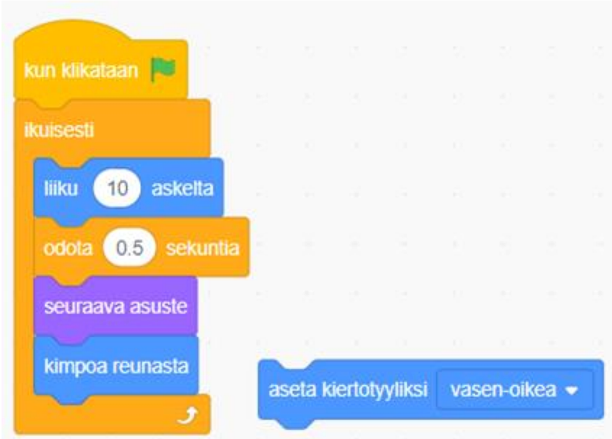


Kuva 41: Erilaisia tapahtumia Scratchissä. Yläreunassa oleva kaari erottaa tapahtumalohkot muista lohkoista. Tapahtumat aloittavat aina jonkin koodin suorituksen.

Esimerkki:

Tehtävä: Laita Scratch-kissa kävelemään näyttämöllä.

Vastaus:

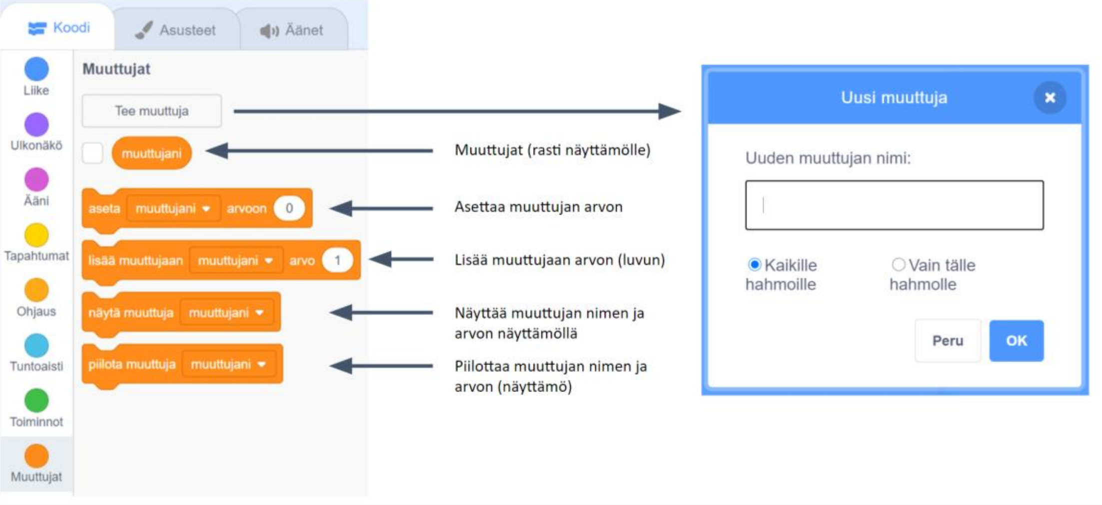


Kissa-hahmon koodivälilehdelle kootaan yllä oleva koodi. Koodin suoritus alkaa, kun ohjelman suoritus alkaa ja toistuu ikuisesti, eli niin kauan kuin ohjelma on käynnissä. Kissa liikkuu 10 askelta ja odottaa hetken. Sitten vaihdetaan kissan kuvaksi animaation seuraava kuva, jossa jalat ovat eri asennossa. Tämä tapahtuu vaihtamalla kissan asustetta.

Ohjelma toimii näinkin, mutta ilman **kimpoa reunasta** –lohkoa kissa kävelee pois näyttämöltä. Kimpoamisen tyyliksi kannattaa vielä lisätä aseta kiertotyyliksi vasenoikea –lohko. Lohko lisätään heti aloituslohkon alle.

6.3.4 TYYPIT, MUUTTUJAT JA LISTAT

Scratchin tyyppejä ovat luvut, merkkijonot ja totuusarvot. Lukuja ja merkkijonoja voi kirjoittaa, mutta totuusarvoja käytetään vain vertailu- ja loogisten operaattoreiden kautta (ks. edellä Kuva 37). Muuttujissa säilytetään tietoa eli dataa. Scratchissä muuttujiin voi tallentaa kahden tyyppisiä arvoja: lukuja tai merkkijonoja (ks. Kuvat 42–44).



Kuva 42: Muuttuja luodaan Muuttujat-lohkorymästä.

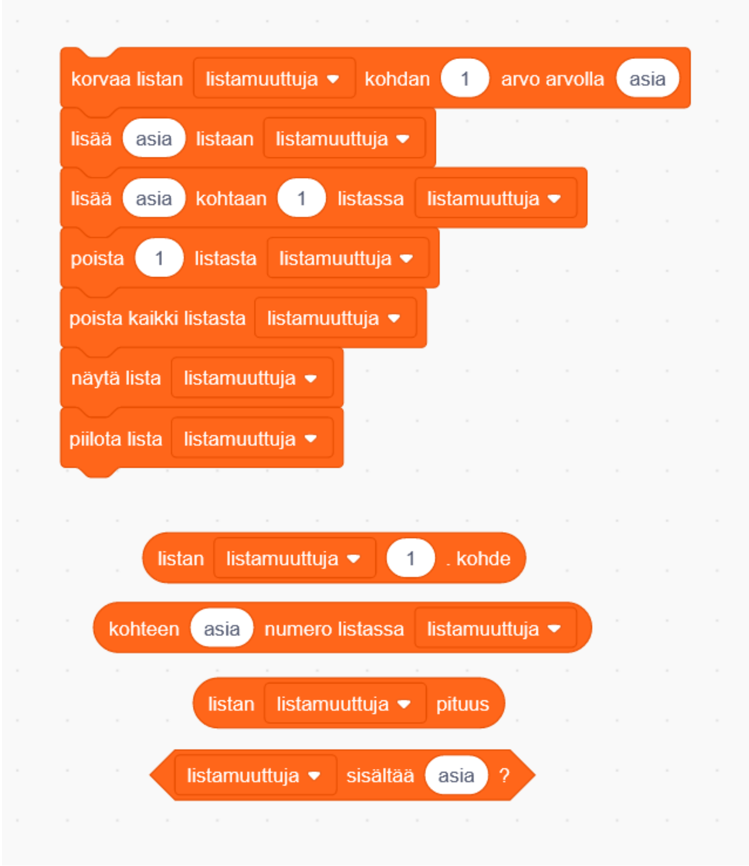
Kuva 43: Scratchissä muuttujiin voi tallentaa lukuja tai merkkijonoja. Kun muuttuja on luotu, voidaan käyttää muuttujille suunnattuja lausekkeita.

Kuva 44: Lisää -lohko kasvattaa arvoa annetulla luvulla. Lohko toimii vain luvuilla: olemassa oleva arvo luetaan ja siihen lisätään annettu arvo, minkä jälkeen uusi arvo tallennetaan muuttujaan.

Kun luodaan muuttuja, on sen vieressä rasti, jolloin se näkyy näyttämöllä. Jälkeenpäin sen näkyvyyttä voi muuttaa ohjelmassa erillisillä lauseilla.

Muuttujiin voi tavallisesti tallentaa vain yhden arvon, koska muuttujan tyyppi on sellainen (luku/merkkijono). Kun on tarve tallentaa useampia arvoja, muuttujaan voidaan tallentaa myös lista, jonka alkiot sisältävät halutut arvot. Scratchissä listamuuttujaan voi lisätä alkiota, siitä voidaan poistaa alkioita ja alkioiden arvoja voi muuttaa. Alkioihin voidaan myös viitata järjestysnumerolla eli **indeksillä**. Listan

ensimmäisen alkion indeksi on 1. Kaikki listaoperaatiot ja -lauseet näkyvät Kuvassa 45 ja 46. Esimerkki listan luomisesta Kuvassa 47.



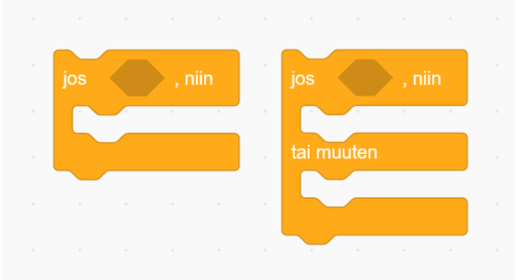
Kuva 45: Scratchissä listamuuttujilla on omat lausekkeensa.

Kuva 46: Pääsy alkioiden sisältöön ja listan tietoihin tapahtuu listalohkojen avulla.

Kuva 47: Esimerkki listojen käytöstä: On luotu listamuuttuja iät, jonne tallennetaan 10 satunnaista lukua väliltä 7–12.

6.3.5 EHTOLAUSEET

Scratchissä on kaksi erilaista ehtolauselohkoa (ks. Kuva 48). Loogisista tai vertailuoperaattoreista koostuvat lausekkeet toimivat ehtona, jonka perusteella joko suoritetaan tai ei suoriteta koodia (Kuvan 48 vasemmanpuoleinen ehtolauselohko) tai suoritetaan ensimmäinen tai toinen koodi (Kuvan 48 oikeanpuoleinen ehtolauselohko).



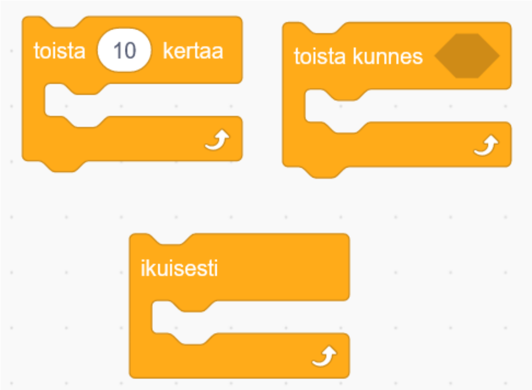
Kuva 48: Scratchissä ehtolauseet ovat valmiita lohkoja, joiden sisälle rakennetaan ehdon perusteella suoritettavat koodit. Vasemmalla olevaa ehtolause suorittaa lohkon tai joukon lohkoja, jos ehto on tosi. Oikealla oleva ehtolause tarjoaa kaksi vaihtoehtoa, jos ehto on tosi suorittaa ensimmäisen joukon lohkoja ja jos epätosi niin toisen.

Ehtolauseisiin voi laittaa minkä tahansa vertailu- tai loogisen operaattorin sisältävän (tai niitä yhdistelevän) lausekkeen (ks. Kuva 49).

Kuva 49: Ehtolauselohkojen ehdot koostuvat vertailu- ja loogisista lausekkeista. Kuvassa pyydetään käyttäjää antamaan jokin luku (tallennetaan vastaus-muuttujaan). Lukua vertaillaan kahden vertailuoperaattorin avulla lukuihin 5 ja 10, jotka on sitten yhdistetty loogisella operaattorilla ja. Selkokielisenä ehtolause on siis "JOS luku on pienempi kuin 5 TAI suurempi kuin 10, NIIN sano Luku on 5-10! 2 sekuntia.".

6.3.6 SILMUKAT

Scratchistä löytyy kolme erilaista silmukkaa: "toista kunnes…", "toista … kertaa" ja "ikuisesti" (ks. Kuva 50). Ensimmäisessä vaihtoehdossa silmukkaa toistetaan niin kauan, kunnes määritelty ehto muuttuu todeksi. Toisessa vaihtoehdossa taas silmukkaa toistetaan silmukassa määritellyn lukumäärän verran. Kolmatta silmukkaa, suoritetaan niin kauan, kun ohjelma on käynnissä tai suoritus lopetetaan toisessa koodipätkässä pysäytä-lohkolla.



Kuva 50: Scratchissä on kolme silmukkarakennetta. Ensimmäistä toistetaan aina tietty lukumäärä, toista niin kauan kun ehto on voimassa ja kolmatta ikuisesti (kunnes ohjelman suoritus loppuu) tai silmukan suoritus lopetetaan toisaalla "Pysäytä"-lohkolla.

Taulukko 5: Erilaisia silmukoita ja muuttujan arvo silmukan suorituksen jälkeen.

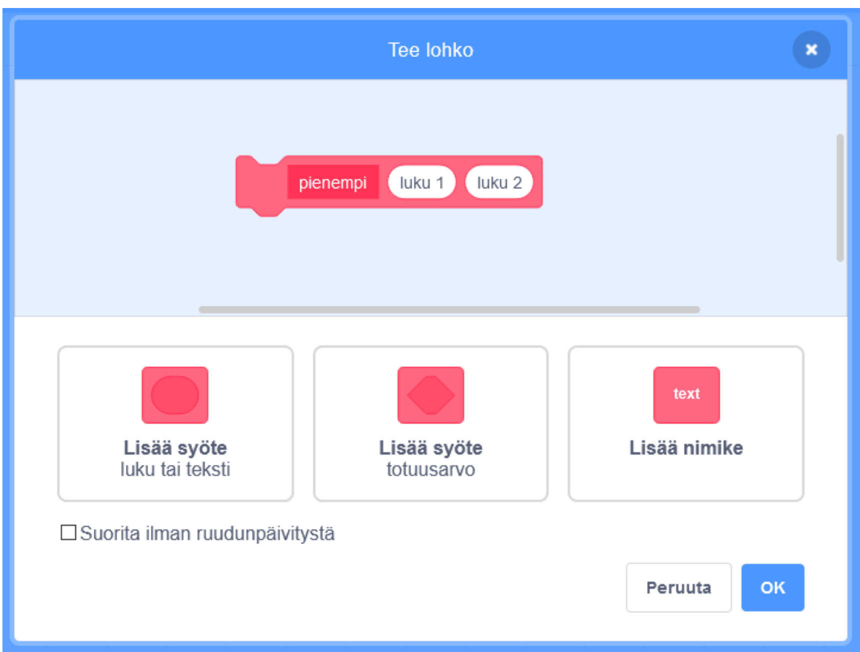
ohjelmakoodi ja selitys	pisteet- muuttujan arvo suorituksen jälkeen
aseta pisteet arvoon 0 toista 10 kertaa lisää muuttujaan pisteet arvo 1 silmukka suoritetaan 10 kertaa	pisteet 10
aseta pisteet arvoon 0 toista kunnes pisteet = 5 lisää muuttujaan pisteet arvo 1 silmukka suoritetaan, kunnes muuttujan pisteet arvo on 5	pisteet 5
aseta pisteet arvoon 0 ikuisesti lisää muuttujaan pisteet arvo 1 ikuinen silmukka	pisteet 9007199254740992

6.4 OHJELMOINNIN KONSEPTIT JATKUVAT

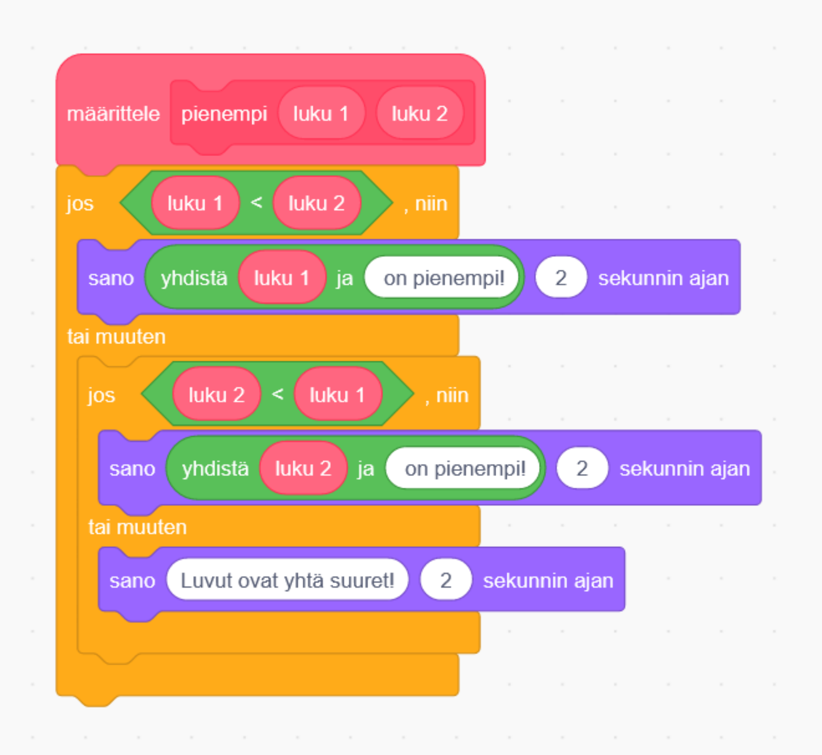
6.4.1 ALIOHJELMAT

Yksi ohjelmoinnin keskeisimmistä edistyneemmistä konsepteista on **aliohjelma**. Aliohjelmien avulla ohjelmakoodista voi tehdä luettavampaa ja koodin pituutta voidaan lyhentää, koska aliohjelmaa voidaan kutsua niin monta kertaa kuin on tarve, eikä aliohjelman sisältämää koodia tarvitse siis kirjoittaa uudelleen. Aliohjelmat vastaavat matemaattisia funktioita: aliohjelmaa kutsuttaessa sille annetaan parametreja, aliohjelman suorituksessa tapahtuu jotain ja lopulta aliohjelma voi palauttaa jonkin arvon kutsukohtaan.

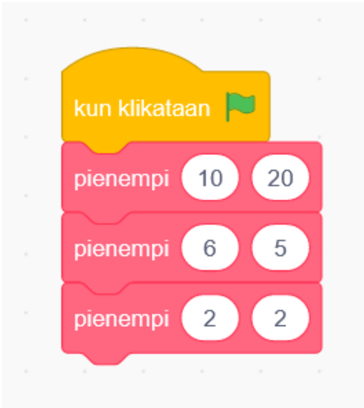
Scratchissä aliohjelmien idea on toteutettu omien lohkojen muodostamisen avulla. Lohkolistasta valitaan kategoria Lohkoni ja Tee lohko. Tämän jälkeen lohkolle valitaan nimi ja parametrit. Parametreja voi olla useampi (ks. kuva 51). Tämän jälkeen klikataan ok, jolloin koodivälilehdelle ilmestyy lohko, jonka perään aliohjelma rakennetaan (ks. kuva 52). Oman lohkon määrittelyn jälkeen sitä voidaan käyttää koodissa niin monta kertaa kuin tarvitaan (ks. kuva 53).



Kuva 51: Scratchissä parametreja lisätään aliohjelmille eli omille lohkoille klikkaamalla halutun tyyppistä parametria.



Kuva 52: Oma lohko muodostetaan määrittele lohko -palikan alle.

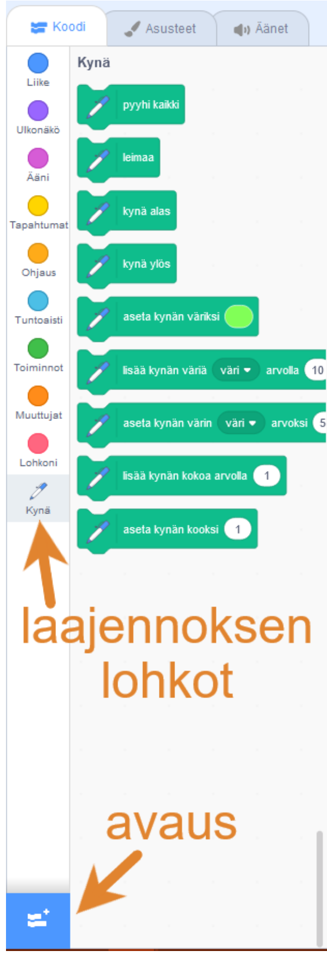


Kuva 53: Omaa lohkoa voidaan käyttää muissa koodipätkissä niin monta kertaa kuin tarvitaan.

6.4.2 KIRJASTOT

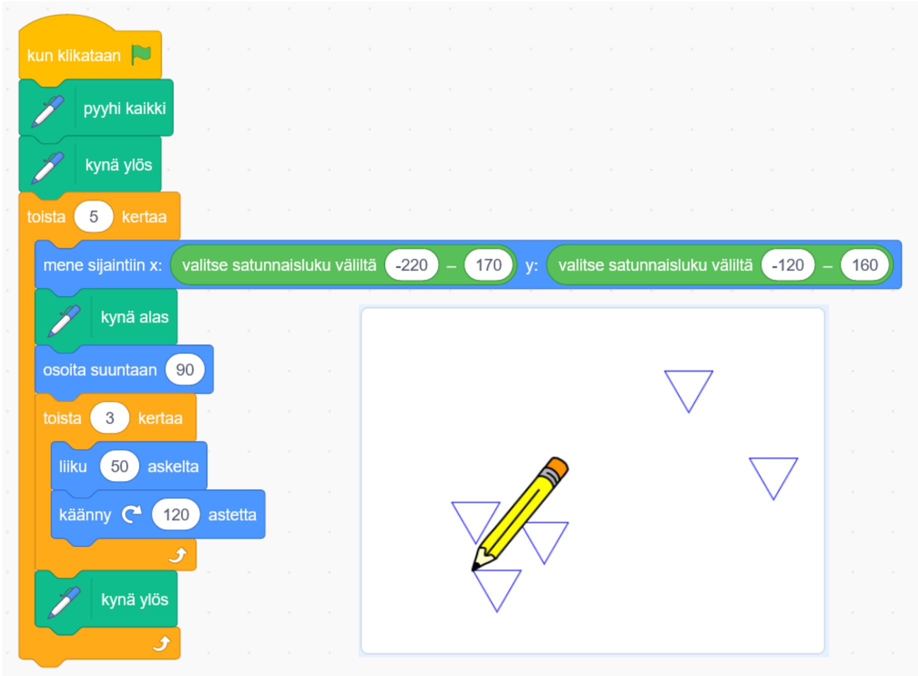
Kirjastot ovat ohjelmaan lisättäviä paketteja, jotka sisältävät valmiiksi toteutettuja ominaisuuksia, joita voidaan kirjaston käyttöönoton jälkeen hyödyntää ohjelmassa. Kirjastojen avulla voidaan selkeyttää ja lyhentää ohjelmaan vaadittua koodia, kun joitain ominaisuuksia saa valmiina ja ohjelmakoodissa voi vain kutsua kyseistä ominaisuutta sen sijaan, että toteuttaisi sen itse.

Scratchissä kirjastoja kutsutaan laajennuksiksi, jotka löytyvät lohkolistan vasemmasta alakulmasta. Kun laajennus on valittu, ilmestyy lohkolistaan uusi kategoria (valittu laajennus), jonka sisältä löytyy laajennuksen lohkot (ks. kuva 54).

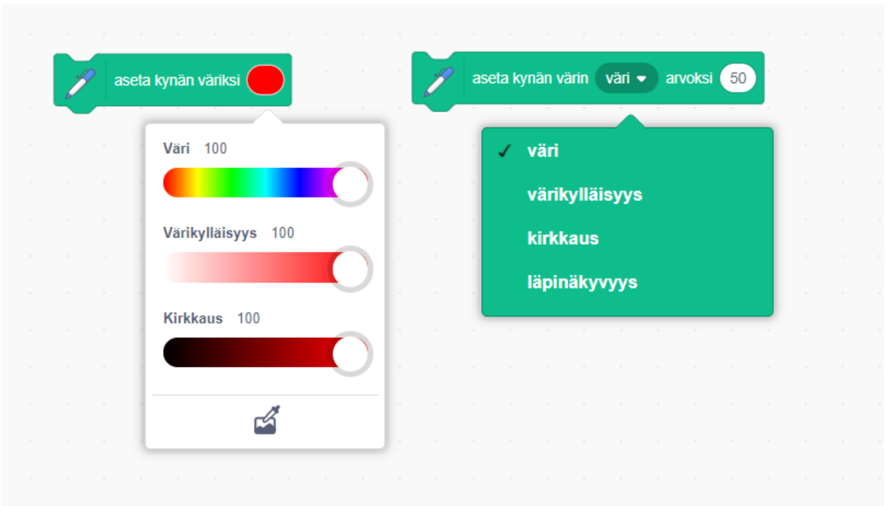


Kuva 54: Kynä-laajennuksen lohkot löytyvät lohkolistasta, kun laajennus on otettu käyttöön.

Yksi Scratchin laajennuksista on Kynä, jonka avulla hahmot voivat liikkuessaan piirtää viivaa näyttämölle (ks. Kuva 55 ja 56).



Kuva 55: Kynä-laajennuksen avulla Scratchin hahmot voivat piirtää viivoja ruudulle. Tässä ohjelman kynähahmo piirtää viisi kolmiota satunnaisiin kohtiin ruutua, kun ohjelma suoritetaan.



Kuva 56: Kynän väriä voi säätää omilla lohkoillaan. Huomaa että jos värin kirkkaus on 0, niin värin muuttaminen ei muuta kynän väriä.

6.5 OHJELMOINNIN SOVELTAMINEN MATEMATIIKASSA

ScratchMaths-projekti yhdistää matematiikkaa, ohjelmoinnillista ajattelua ja ohjelmointia (https://www.ucl.ac.uk/ioe/research/projects/ucl-scratchmaths). ScratchMaths-opetussuunnitelmassa on yli 40 tuntia materiaalia englanniksi: kalvot, tehtäväpohjat Scratchiin ja opettajan oppaita.

Tehtäväpohjat toimivat suomeksikin hyvin pienin muutoksin, koska lohkot Scratch suomentaa automaattisesti, kun vaihtaa projektin kielen. Sen jälkeen täytyy enää suomentaa mm. muuttujien ja omien lohkojen nimet.

ScratchMaths-tehtävissä on sekä tietokoneella (Scratchissä) tehtäviä että ilman tietokonetta tehtäviä harjoituksia. Moduulit koostuvat tutkimuksista (investigations), jotka koskevat eri aihealueita. Tutkimusten sisällä on samaan aihealueeseen liittyviä pienempiä tehtäviä (activity). Jotkin tehtävistä vaativat, että toinen tehtävä on tehty ensin, mutta muuten niistä voi valita haluamansa.

Tehtävien on tarkoitus opettaa matematiikkaa ohjelmoinnin avulla. Opettajan materiaaleissa on ohjeita sekä tehtävänantoja varten, että tehtävän yhdistämiseen matematiikkaan. ScratchMaths-tehtävissä korostuu matematiikan aiheet. ScratchMath-projektissa tehdyn tutkimuksen mukaan oppilaat oppivat ohjelmointia paremmin, kun tehtävät olivat yhdistetty matematiikkaan. Matematiikan oppiminen taas pysyi samalla tasolla, kuin ilman ohjelmointia.

6.5.1 ESIMERKKI 1: PELATAAN PAIKKA-ARVOJEN KANSSA

Moduulin 4 Tutkimus 1: Paikka-arvomallit käsittelee numeroiden paikka-arvoja luvuissa. Tutkimus sisältää neljä tehtävää:

1. Numerot ylös, numerot alas
2. Kääntö kääntö, työntö työntö (harjoitus ilman tietokonetta)
3. Pelataan paikka-arvojen kanssa
4. Jonot (Lisätehtävä)

Tutkimuksen matemaattiset tavoitteet ovat

- paikka-arvoja koskevien luku- ja käytännön ongelmien ratkaiseminen,
- laskeminen päässä (sisältäen laskutoimituksia, joissa on useita eri operaattoreita),
- lineaaristen lukujonojen luominen ja kuvaileminen ja
- työskenteleminen satunnaisuutta sisältävien kokeilujen kanssa.

Tässä esimerkissä keskitytään tehtävään 3: Pelataan paikka-arvojen kanssa, jossa tavoitteena on luoda Scratchillä laskuriohjelma. Tehtävän oppimistavoitteet ovat

- tutkia ja rakentaa paikka-arvomalli neljälle numerolle ja
- selittää, miten neljä numerohahmoa laskevat nollasta 9999:än.

Tehtävän ohjeet	**Yhteys matematiikkaan**
Oppilaat avaavat projektin Scratchissä (materiaalien 41-Digits Up).	
1: Oppilaat vaihtavat toiseen taustaan, jossa on neljä tyhjää paikkaa neljälle numerolle. He	

muokkaavat ykköset-hahmon koodia niin, että se menee oikealle paikalleen oikeanpuolimmaisen paikan päälle.	
2: Oppilaat poistavat kaiken muun ykköset-hahmon koodin paitsi aloituskoodin (kun klikataan vihreää lippua). Sitten he monistavat ykköset-hahmon ja nimeävät uuden hahmon kymmeniksi.	Muistuta oppilaille, että numerot, jotka on laitettu kymmenten paikalle edustavat arvoa numero × 10. Keskustelua: Miksi tarvitaan toinen hahmo, jos halutaan laskea lukuun 99 asti 9 sijaan?
3: Oppilaat muokkaavat kymmenet-hahmon koodia niin, että se menee kymmenten paikalle. He toistavat saman prosessin ja luovat kolmannen ja neljännen hahmon: nimetään sadoiksi ja tuhansiksi ja muokataan niiden koodi, jotta ne menevät oikeille paikoilleen.	
4: Oppilaat rakentavat ykköset-hahmolle yksinkertaisen “kun tätä hahmoa klikataan, seuraava asuste” koodin. Tutkimuksen edellisistä tehtävistä oppilaille olisi jo tuttua se, että kun hahmo saavuttaa viimeisen asusteensa 0, täytyy kymmentenkin muuttua eli ykköset-hahmon täytyy viestittää “lisää 10”.	Keskustelua: Kun lasketaan klikkaamalla ykköset-hahmoa, milloin kymmenet-hahmon pitäisi kasvattaa arvoaan eli vaihtaa seuraava asuste?
5: Kymmenet-hahmon täytyy reagoida “lisää 10” -viestiin vaihtamalla seuraavaan asusteeseen. Oppilaat rakentavat “kun vastaanotan “lisää 10”, seuraava asuste” ja laajentavat sitä lähettämään viestin “lisää 100”, kun kymmenet-hahmo saavuttaa viimeisen asusteensa 0.	Kysy kysymyksiä yhdistämään klikkausten määrä yhteen- ja vähennyslaskuun: Kuinka monta klikkausta tarvitaan, jos näyttämöllä on luku 578 ja halutaan 600? Ykköset-hahmoa klikattiin 17 kertaa, jotta päästiin lukuun 965. Mistä luvusta aloitettiin? Tarkistakaa vastaukset.
6: Oppilaat muokkaavat aloituskoodissa määriteltyä aloitusasustetta, jotta he voivat tutkia tilanteita, joissa aloitusluku onkin suurempi (esim. 6997) klikkaamalla vain muutaman kerran.	
Oppilaat rakentavat sopivat koodit sadat- ja tuhannet-hahmoille. He testaavat uutta neljän numeron lukua.	Oppilaat rakentavat sopivat koodit sadat- ja tuhannet-hahmoille. He testaavat uutta neljän numeron lukua.

Tehtävässä siis harjoitellaan ohjelmointia, mutta siihen yhdistetään matematiikkaa. Tehtävän aihe on jo itsessään matemaattinen, mutta matematiikan oppimista voidaan myös tukea kysymyksillä ja keskustelulla. Ohjelmoinnin oppimisen kannalta on tarkoituksenmukaista, että oppilaat pohtivat sitä, mitä

he tekevät ohjelmoidessaan, eivätkä vain tee jotain ja testaa sitten toimiiko se niin kuin pitäisi. Tämä myös vahvistaa heidän ymmärrystään tehtävässä käytetystä matematiikan aiheesta.

Linkki materiaaliin (englanniksi): https://www.ucl.ac.uk/ioe/research/projects/ucl-scratchmaths/curriculum-materials/module-4-building-numbers

6.5.2 ESIMERKKI 2: PIIRRETÄÄN SÄÄNNÖLLISIÄ MONIKULMIOITA

Moduulin 2 Tutkimus 2: Piirretään monikulmioita keskittyy erilaisiin geometrisiin kuvioihin, niiden ominaisuuksiin ja niiden piirtämiseen ohjelmoimalla.

Tutkimus sisältää neljä tehtävää:

1. Piirretään säännöllisiä monikulmioita
2. Monikulmiokoodeja (harjoitus ilman tietokonetta)
3. Käytetään ja määritellään omia lohkoja
4. Yhdistellään uusia lohkoja

Tutkimuksen matemaattiset tavoitteet ovat

- geometristen kuvioiden vertaileminen ja luokitteleminen,
- tiettyjen pisteiden merkitseminen ja sivujen piirtäminen annetun monikulmion muodostamiseksi,
- suorakulmion ominaisuuksien käyttäminen siihen liittyvien faktojen päättelemiseksi,
- säännöllisten ja epäsäännöllisten monikulmioiden erottaminen perustuen päättelyyn yhtä pitkistä sivuista ja yhtä suurista kulmista,
- kulmien tunnistaminen tietyssä pisteessä,
- kokonainen käännös (360°),
- kulmien tunnistaminen suoralla ja
- puolikas käännös (180°).

Tässä esimerkissä keskitytään tehtävään 1: Piirretään säännöllisiä monikulmioita.

Tehtävän oppimistavoitteet ovat

- tutkia, miten käyttää kynälaajennosta säännöllisten monikulmioiden piirtämiseen ja
- yhdistää tehtävä tietoon erilaisten monikulmioiden ominaisuuksista.

Tehtävän ohjeet	**Yhteys matematiikkaan**
Oppilaat avaavat projektin Scratchissä (materiaalien 22-Drawing Polygons).	

1: Oppilaat suorittavat aloituskoodin, joka laittaa kynän kärjen alas niin, että kovakuoriainen on valmis piirtämään.	
2: Oppilaat yhdistävät yhden liiku-lohkon ja yhden käänny-lohkon koodialueelle.	
3: Oppilaat valitsevat ja asettavat arvot kumpaankin lohkoon ja klikkaavat koodipätkää useaan kertaan (tässä ei siis käytetä silmukkaa).	Oppilaat keksivät erilaisia monikulmioita. Keskustelua: Mitä monikulmioita piirsit? Kuinka monta sivua niissä oli?
4: Opppilaat lisäävät toista … kertaa -lohkon ja asettavat siihen arvoksi pienimmän luvun, jolla heidän valitsemansa monikulmion voi piirtää valmiiksi yhdellä klikkauksella. Jos ensimmäinen lohko silmukassa on käänny-lohko, on paljon vaikeampaa ja hämmentävämpää käyttää sellaista koodia monimutkaisempien kuvioiden piirtämiseen (kuten neliöistä mudostettu torni tai talo), joita tehdään muissa tutkimuksen tehtävissä. Oppilaita kannattaakin kannustaa aloittamaan silmukka liiku-lohkolla.	Keskustelua: Mitä monikulmioita piirsit? Kuinka monta sivua niissä oli? Mistä kovakuoriainen aloitti ja minne se “katsoi”? Minne kovakuoriainen päätyi ja minne se katsoo nyt? Kuinka monta astetta kovakuoriainen kääntyi yhteensä, jotta se sai muodostettua suljetun kuvion? Kuinka monta askelta kovakuoriainen kulki yhteensä?
5: Oppilaat luovat koodin, joka piirtää neliön ja sitten tasasivuisen kolmion.	Keskustelua: Saitko piirrettyä tasasivuisen kolmion? Miten rakensit koodin varmistaaksesi, että kolmio on tasasivuinen? Miksi oli vaikeampaa piirtää kolmio kuin neliö?

Oppilaita voi auttaa monikulmioiden piirtämisessä harjoituksella, joka tehdään ilman tietokonetta.

1. Pyydä oppilasta kuvittelemaan, että hän on kovakuoriainen.
2. Opasta tai pyydä toista oppilasta opastamaan kovakuoriaista kulkemaan tietyn verran askelia ja sitten kääntymään 90 astetta toistuvasti neliön piirtämiseksi.
3. Tehkää sama kolmiolle ja keskustelkaa eri kulmista, joiden verran kovakuoriaisen on käännyttävä kummassakin tilanteessa.

Tämäkin tehtävä on selvästi aiheeltaan matemaattinen, mutta oppilaiden matemaattista ajattelua voi tukea kysymyksillä, pohdinnoilla ja keskustelussa tehtävän ohessa.

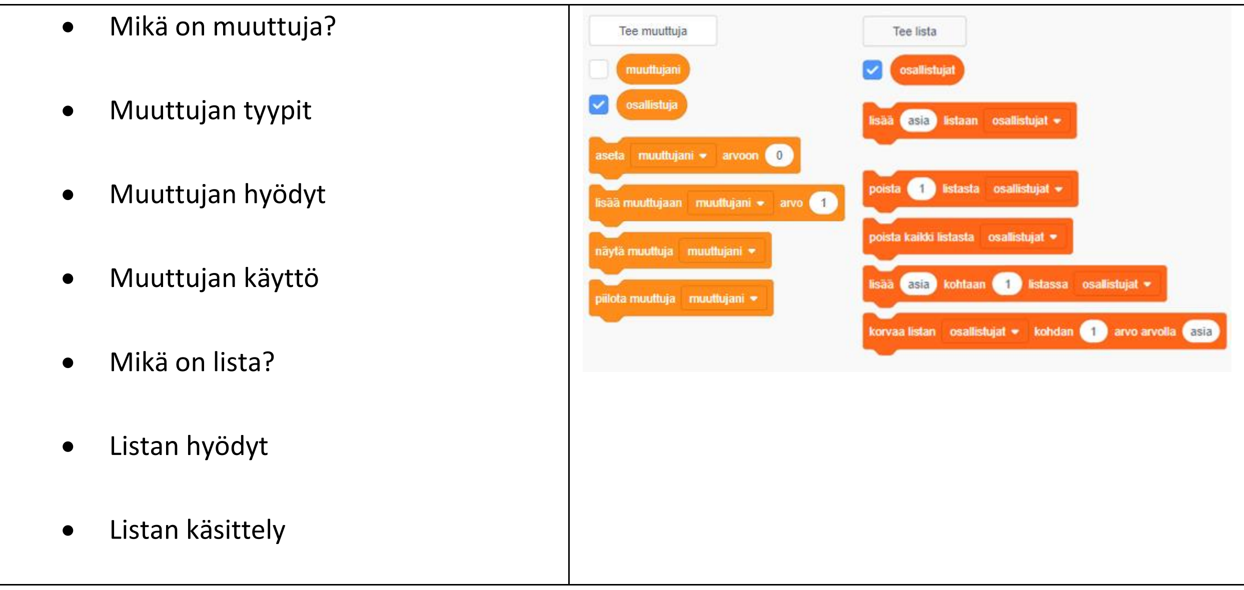
Linkki materiaaliin (englanniksi): https://www.ucl.ac.uk/ioe/research/projects/ucl-scratchmaths/curriculum-materials/module-2-beetle-geometry

7 TYÖPAJA 3: SCRATCH-OHJELMOINTI

Työpajan tavoitteena oli käydä lävitse ohjelmoinnin perusteet Scratch-ohjelmointiympäristössä. Muuttujat ovat keskeisiä, niiden avulla tietoa tallennetaan muistiin, tietoa palautetaan muistista ja tietoa voidaan käsitellä. Muuttujat sisältävät yleensä yhden arvon, joka on jotain tyyppiä kuten numero, teksti, totuusarvo jne. Kun tarvitaan useampien arvojen tallentamista, käytetään tietotyyppinä tietorakennetta kuten listaa, joka mahdollistaa useamman arvon tallentamisen yhtenä kokonaisuutena. Ehtolauseet ja silmukat ohjaavat ohjelman suoritusta. Ehtolause tarjoaa vaihtoehtoisen suorituspolun, jos jokin ennalta määritelty ehto täyttyy. Silmukat toistavat joukon lauseita, niin kauan, kun määritelty ehto on voimassa. Aliohjelmilla voidaan tehdä ohjelman koodista hallittavampaa, pilkkomalla se itsenäisiin osiin. Kirjastot tarjoavat uusia ominaisuuksia, joita voi hyödyntää omissa ohjelmissa.

Työpaja alkoi lyhyellä alustuksella, jossa esitettiin yhteenveto edeltävästä verkkosisällöstä ja käytiin lävitse edellisen työpajan kotitehtävien vastaukset. Alustusta seurasi kolme erillistä osiota, joissa käsiteltiin muuttujia ja listoja, ehtolauseiden ja silmukoiden muodostamista, sekä aliohjelmien luomista ja kirjastojen ominaisuuksien hyödyntämistä.

Muuttujat ja listat

• Mikä on muuttuja? • Muuttujan tyypit • Muuttujan hyödyt • Muuttujan käyttö • Mikä on lista? • Listan hyödyt • Listan käsittely	

Ehtolauseet ja silmukat

<table>
<tr>
<td>

- Ehtolauseen käyttö
- Ehtolauseen rakenne
- Logiikka ja vertailuoperaattorit
- Toistolauseen (silmukoiden) käyttö
- Toistolauseen tyypit
- Toistolauseen ohjaaminen
- Ikuiset silmukat

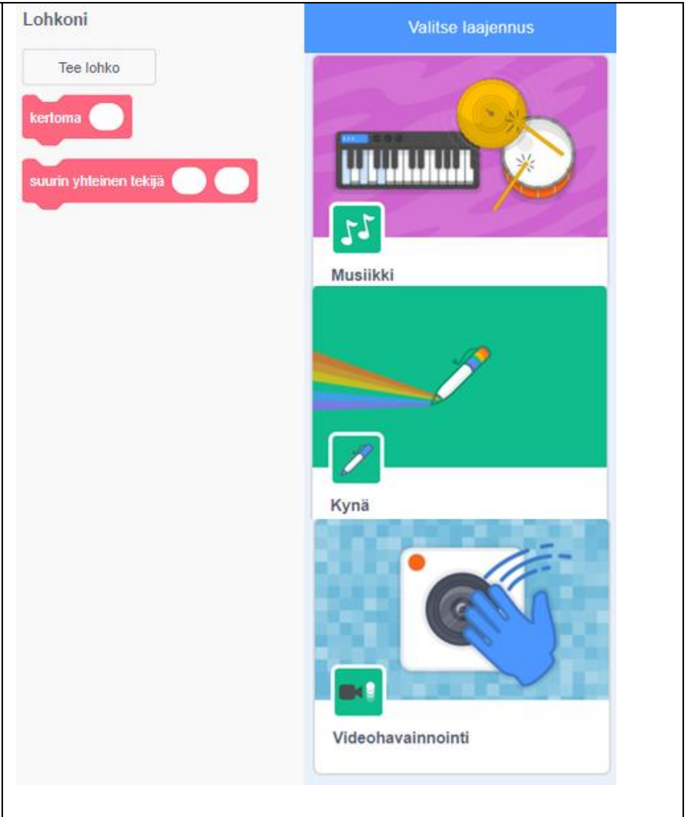
</td>
<td>

</td>
</tr>
</table>

Aliohjelmat ja kirjastot

<table>
<tr>
<td>

- Aliohjelmat eli omat lohkot
- Mitä hyötyä aliohjelmista
- Aliohjelmat ja syötteet
- Kirjastot eli laajennukset
- Käytön mahdollistavat uudet lohkot
- Lisäominaisuudet
- Laitteiden ohjaaminen
- Ulkoiset palvelut

</td>
<td>


</td>
</tr>
</table>

8 VERKKOSISÄLTÖ 4: OHJELMOINNIN OPETUKSEN MENETELMIÄ

Aiemmin on käsitelty ohjelmoinnillista ajattelua, ohjelmointia ja niiden yhdistämistä matematiikkaan. Näillä on luotu alakoulun ohjelmoinnin opetuksessa tarvittavaa tietopohjaa.

Tässä osiossa käsitellään ohjelmoinnin opetusta. Kun ohjelmointia opetetaan, on tärkeää kiinnittää huomiota siihen, että oppilaat ymmärtävät, mitä ohjelmassa tapahtuu. Vaikka ohjelma toimisikin tehtävänannossa kuvatusti, voi silti ohjelman ymmärrys (*program comprehension*) jäädä vaillinaiseksi. Ohjelman ymmärryksellä tarkoitetaan prosessia, jossa oppilas muodostaa oman mentaalisen mallinsa ohjelmasta. Ymmärrys on tärkeää, jotta oppilaat voivat lukea ja tulkita muiden luomaa koodia ja reflektoida omaansa. Nämä taidot ovat tarpeellisia, kun kirjoitetaan ja laajennetaan koodia tai etsitään mahdollisia virheitä koodista. Toimivat ohjelmat antavat oppilaille onnistumisen tunteita ja motivoivat heitä, mutta ohjelman ymmärrystä täytyy harjoitella ja testata toisenlaisilla tehtävillä.

Ensimmäiseksi ohjelmoinnin opetuksen menetelmistä käsitellään unplugged-harjoitusten roolia alakoulun ohjelmoinnissa. Tämän jälkeen esitellään TIPP&SEE ja 5E-menetelmät, jotka on suunniteltu ohjelmoinnin opetusta varten. Lopuksi käydään läpi erilaisia tehtävätyyppejä, joita ohjelmoinnin opetuksessa voi hyödyntää.

8.1 ENSIN UNPLUGGED, SITTEN OHJELMOINTI

Tässä kappaleessa pohditaan toiminnallisten harjoitusten roolia alakoulun ohjelmoinnin ja ohjelmoinnillisen ajattelun opetuksessa. Aiemmasta muistamme, että opetussuunnitelman mukaan toiminnallisia harjoituksia käytetään erityisesti alkuopetuksessa.

Unplugged-termi viittaa ohjelmoinnin ja erityisesti ohjelmoinnillisen ajattelun harjoitteluun toiminnallisesti, ilman tietokonetta. Termi tuli käyttöön CS Unplugged-tehtäväkokoelman myötä jo parikymmentä vuotta sitten. Sen jälkeen unplugged-lähestymistapaa on tutkittu paljon. Viime vuosien aikana on ilmestynyt useita julkaisuja, joissa tutkitaan missä määrin ja miten ohjelmoinnillista ajattelua opitaan toiminnallisilla harjoituksilla. On havaittu, että toiminnalliset harjoitukset vahvistavat ohjelmoinnillisen ajattelun käsitteiden oppimista. [1]

Unplugged-harjoitukset liittyvät vahvasti kasvatustieteiden nykyiseen konstruktivistiseen suuntaukseen. Alakoululaisten kanssa on luontevaa lähestyä opetettavaa asiaa ensin konkreettisten esimerkkien kautta, itse kokeillen, ja edetä myöhemmin abstraktimpaan suuntaan. Kun esimerkiksi algoritmin toimintaa mallinnetaan ensin konkreettisesti toiminnallisin menetelmin, voidaan varmistua siitä, että toiminta ymmärretään ennen kuin siirrytään ohjelmointiympäristön ääreen. Lisäksi toiminnalliset, oppilaan kokemusmaailmaan linkittyvät harjoitukset tuovat esille sen, että tarkoitus on oppia ajattelutaitoja, joita voi käyttää muutenkin kuin ohjelmoinnissa.

Alkuperäisissä CS Unplugged-tehtävissä ei ehdotettu materiaalin yhdistämistä tietokoneella tehtäviin harjoituksiin. Muutenkaan tehtäväkokoelmaa ei alunperin julkaistu kokonaisena opetussuunnitelmana, vaan tarkoitus oli herättää tehtäviä käyttävien oppilaiden ja opiskelijoiden kiinnostus tietojenkäsittelytieteisiin. CS Unplugged-tehtävien laatijat linjaavat harjoitusten pääperiaatteita seuraavasti:

- ei käytetä tietokonetta eikä mitään ohjelmointikieltä,
- tehtävissä on leikinomaisuutta tai oppilas haastetaan tutkimaan uutta asiaa,
- tehtävät ovat kinesteettisiä,
- lähestymistapa on konstruktivistinen,
- ohjeet ovat lyhyet ja yksinkertaiset ja
- tehtävissä on tarinantuntua.

Näitä suuntaviivoja seuraamalla voi itsekin laatia toiminnallisia harjoituksia ohjelmoinnin ja ohjelmoinnillisen ajattelun opetuksen tueksi. [2]

Kinesteettinen oppiminen: oppiminen toiminnan tai liikeaistin kautta.

Konstruktivistinen oppiminen: oppiminen on aktiivista uusien kokemusten ja tietojen sovittamista aiempiin tietorakennelmiin.

Tutkijoita on kiinnostanut myös toiminnallisten harjoitusten liittyminen laajemmin ohjelmoinnin ja ohjelmoinnillisen ajattelun opetussuunnitelmiin. Kotsopoulos ehdottaa nelivaiheista opetusmenetelmää, joka koostuu toiminnallisesta, näpertely-, uuden luomis- ja miksausosuudesta. Kotsopoulos työryhmineen totesi, että kaikki vaiheet ovat tarpeellisia ohjelmoinnillisen ajattelun oppimiseksi, ja erityisesti aloittelijoille yllä lueteltu järjestys on tärkeä. Vaiheet liittyvät osittain oppilaan kehitykselliseen jatkumoon. Oppilaat työskentelevät kaikissa vaiheissa lähikehityksen vyöhykkeellään, mutta haasteet kasvavat siirryttäessä vaiheesta toiseen. Jokaisessa vaiheessa opitaan uutta. Tämän kappaleen kannalta on mielenkiintoista, että myös tässä menetelmässä toiminnalliset-, unplugged-harjoitukset edeltävät varsinaista ohjelmointia. Unplugged-harjoituksiin ei tarvita teknistä osaamista ja ne ovat usein yhteistoiminnallisia ja kinesteettisiä sekä kasvattavat oppimismotivaatiota. [3]

Jos ajatellaan nimenomaan suomalaista opetussuunnitelmaa, jossa ohjelmointi ja ohjelmoinnillinen ajattelu on integroitu matematiikan opetukseen, tuntuu luontevalta aloittaa uuden asian opettelu molempia oppiaineita integroivalla toiminnallisella harjoituksella. Samaan tulokseen tuli Benton työryhmineen tutkiessaan matematiikan ja ohjelmoinnin integrointia. Tutkimusryhmä löysi toisiaan tukevia matematiikan ja ohjelmoinnin aihealueita ja ehdotti matematiikan opetussuunnitelman spiraaliluonteen huomioivaa opetusmenetelmää, jossa edetään toiminnallisista harjoituksista kohti varsinaisia ohjelmointitehtäviä. [4]

<u>Lähteet:</u>

[1] Huang, Looi: A critical review of literature on “unplugged” pedagogies in K-12 computer science and computational thinking education

[2] Bell, Vahrenhold: CS Unplugged—How Is It Used, and Does It Work?

[3] Kotsopoulos et al.: A Pedagogical Framework for Computational Thinking

[4] Benton et al.: Bridging Primary Programming and Mathematics: Some Findings of Design Research in England

8.1.1 ESIMERKKINÄ SCRATCHMATHS

ScratchMaths-materiaalin opettajan ohjeissa on vinkkejä erilaisiin unplugged-tehtäviin, jotka liittyvät kiinteästi materiaalissa esiteltyihin ohjelmointitehtäviin. Jokaiseen kuuteen moduuliin kuuluu useita unplugged-tehtäviä. Osa tehtävistä on oppilaille jaettavia monisteita ja osa tyypillisempiä unplugged-tehtäviä, joissa hyödynnetään konstruktivistista ja kinesteettista lähestymistapa. Voit ladata ScratchMaths-materiaalin täydellisenä osoitteesta https://www.ucl.ac.uk/ioe/research/projects/ucl-scratchmaths/ucl-scratchmaths-curriculum. Tässä esitellään kaksi tehtävää.

8.1.1.1 MINÄ ÖTÖKKÄNÄ

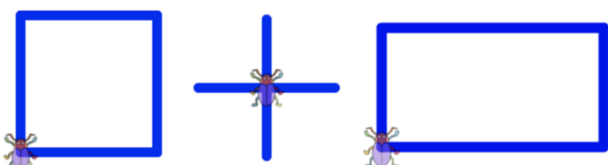
ScratchMaths-materiaalin toisessa moduulissa tutkitaan monikulmioiden piirtämistä ohjelmoimalla. Moduulin nimi on Ötökkägeometriaa (*Beetle Geometry*). Harjoituksen nimi on Minä ötökkänä (*I am Beetle*) ja täydelliset ohjeet löytyvät opettajan materiaalien sivulta 14.

Edustaa tehtävätyyppejä (ks. kappale 4. 5E-menetelmä matematiikan ja ohjelmoinnin yhdistämiseen):

- Explore - Tutkia
- Exchange - Tehdä yhteistyötä

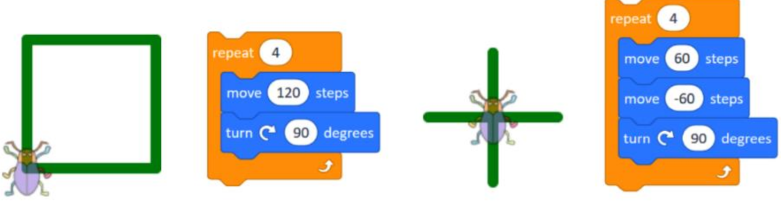
<u>Ohjeet (tiivistelmä):</u>

Toimitaan pareittain. Toinen oppilas valitsee yhden annetuista kuvioista ja ohjaa parinsa liikkumaan lattialla kuvion mukaisesti. Lattialla liikkunut oppilas piirtää paperille liikkeidensä muodostaman kuvion. Verrataan alkuperäistä ja piirrettyä.



Plugged-osio:

Tehtävää voi jatkaa Scratchissa. Oppilaat ohjelmoivat neliön tai ristin piirtävän ohjelman ja sen jälkeen auttavat pariaan tekemään vastaavan ohjelman. Molemmat **oppilaat pääsevät selittämään koodinsa toimintaa**. Oman koodin toiminnan selittäminen on tehokas tapa syventää ohjelmointikäsitteen osaamista.



8.1.1.2 LÄHETÄ JA VASTAANOTA

Kolmannessa moduulissa Hahmojen vuorovaikutus (*Interacting Sprites*) aiheena on hahmojen animointi viestien lähettämisen avulla. Harjoituksen nimi on Lähetä ja vastaanota (*Broadcast and receive*) ja se on opettajan materiaalissa sivulla 35.

Edustaa tehtävätyyppejä:

- Explore - Tutkia
- Explain - Kertoa
- Envisage - Kuvitella

<u>**Ohjeet (tiivistelmä):**</u>

Tehtävään tarvitaan loru, jonka säkeet jaetaan korteille. Jokaiseen korttiin (paitsi aloituskorttiin) tulee ohjeet oman säkeen sanomiseen:

Kun kuulet <edeltävä säe>

Sano <oma säe>

Osa säkeistä on varattu yhden oppilaan lausumiksi, osan toistaa isompi ryhmä kuorossa. Jokainen oppilas saa oman kortin.

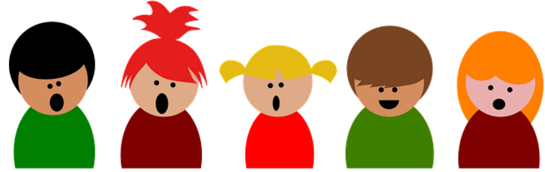
1. The Grand old Duke of York he had ten thousand men
2. He marched them up to the top of the hill
3. And he marched them down again
4. When they were up -> kuoro A) They were up
5. And when they were down -> kuoro B) They were up

6. And when they were only halfway up -> kuoro C) They were neither up nor down

Suomenkielisille oppilaille valitaan suomenkielinen loru. Kannattaa valita jotakin joka puhuttelee omaa ryhmää. Älä valitse sellaista lorua, johon sisältyy ikuinen silmukka :)

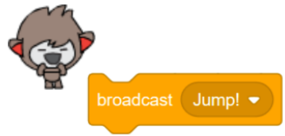
1. Enten tenten teelikamenten
2. Hissun kissun -> kuoro A) vaapula vissun
3. Eelin keelin klot -> kuoro B) viipula vaapula vot
4. Eskon saun -> kuoro C) piun paun
5. Nyt mä lähden tästä pelistä pois
6. Puh pah pelistä pois

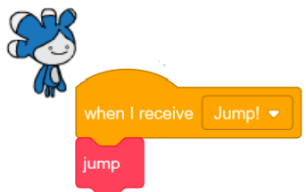
Taiteellisesti esitetyn lorun jälkeen mietitään miten oppilas tiesi milloin oma lause pitää sanoa (kuuntelu), ketkä kuulivat lorun lauseet (kaikki) ja mitä tapahtui, kun usealla oppilaalla oli sama ohje.



Plugged-osio:

Moduulissa jatketaan käsittelemällä Scratchin Lähetä viesti -toiminto. **Aiemmin aloitettua** Scratch-hahmojen animointitehtävää **laajennetaan** lisäämällä hahmon reagointi kuulemaansa viestiin. Aiemmin luodun koodin laajentaminen uudella ominaisuudella liittää uuden asian tiiviisti jo opittuihin ja antaa sille heti käyttötarkoituksen.





8.2 TIPP&SEE MENETELMÄ OHJELMOINNIN OPPIMISEN TUKEMISEEN

Ohjelmoinnin tuominen yleiseksi oppiaineeksi peruskoulussa asettaa haasteita. Vapaaehtoisen ohjelmoinnin opetuksen (koodauskerhot ym.) etuna on, että osallistujat ovat motivoituneita ja usein

omaavat harrastuksen kautta saadun pohjatiedon. Koulussa tällaista asennetta tai taustatietoa ei voi olettaa.

Erilaiset oppimisvaikeudet vaikuttavat myös ohjelmoinnin oppimiseen. Suomessa yhteys matematiikkaan voi antaa vaikutelman ohjelmoinnista vaikeana aiheena ja voidaan myös ajatella, että siihen käytetty aika on pois matematiikan oppimiselta. Erään opettajan kommentti, jossa todettiin ohjelmoinnin pilaavan kaksi ainetta, kuvastaa tätä ajatusta.

Ohjelmointia voidaan käyttää myös matematiikan oppimisen välineenä. Lisäksi on kehitetty menetelmiä ohjelmoinnin oppimisen edistämiseen. Keskitymme tässä oppimistrategiaan ja sitä hyödyntävään menetelmään, jonka tavoitteena on ohjelmoinnin oppimisen aloittamisen tukeminen. Kokeile -> Muokkaa -> Luo -strategia on luotu ohjelmoinnin oppimisen kynnyksen madaltamiseen.

Perinteisesti ohjelmoinnin opetuksessa esitellään uusi käsite sitä hyödyntävän esimerkin avulla. Tämän jälkeen oppilaat harjoittelevat sen käyttämistä erilaisilla tehtävillä. Ohjelmoinnissa haasteena on uuden käsitteen lisäksi koko tehtävän ratkaisevan koodin luonti. Antamalla oppilaille esimerkkikoodi tutkittavaksi, jossa uutta käsitettä on käytetty, ja tämän jälkeen tehtäväksi esimerkin muokkaaminen, poistaa uuden luomisen haasteen. Kokeile -> Muokkaa vaiheen jälkeen voidaan siirtyä ratkomaan vastaavanlaisia tehtäviä soveltaen esimerkin mukaista koodia.

TIPP&SEE on menetelmä Kokeile -> Muokkaa strategian toteuttamiseen Scratch-kielen esimerkeillä (ks. Taulukko 6), joka hyödyntää myös Scratch-sivuston tapaa esittää ohjelmointiprojekteja. Oppimisstrategia on jaettu kahteen osaan kokeilemisen (TIPP) ja muokkaamiseen (SEE). Kokeilu aloitetaan lukemalla otsikko, jotta saadaan kuva projektista ja selvittämällä millaisia ohjeita sen hyödyntämiseen on annettu. Sitten mietitään mikä on projektin tavoite ja mitä koodista voi oppia. Lopuksi kokeillaan projektia käynnistämällä se vihreästä lipusta ja katsotaan mitä tapahtuu. Tärkeätä on kiinnittää huomiota hahmojen toimintaan.

Taulukko 6: TIPP&SEE menetelmän ohjeet Scratch-projektista oppimiseen

TIPP	
Otsikko (Title)	Mikä on projektin otsikko? Mitä se kertoo projektista?
Ohjeet (Instructions)	Mitä ohjeita on annettu projektin hyödyntämiseksi / tekemiseksi?
Tavoite (Purpose)	Mikä on projektin tavoite? Mitä voit oppia sen koodista?

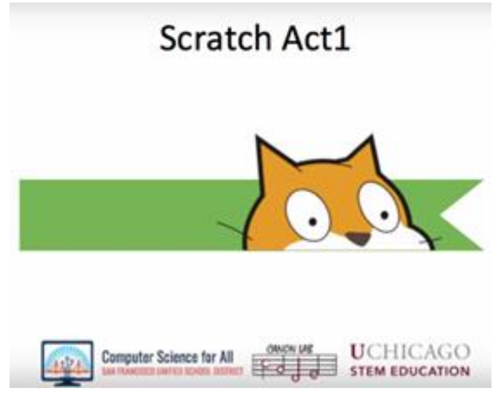
Kokeile (Play)	Kokeile projektia suorittamalla se ja katsomalla mitä tapahtuu! Mitä hahmot tekevät?
SEE	
Hahmot (Sprites)	Valitse hahmo, josta haluat oppia tai johon haluat tehdä muutoksia.
Tapahtumat (Events)	Katso mikä käynnistää koodipätkät. Mitkä koodipätkät ovat hyödyllisimpiä?
Muokkaa (Explore)	Kokeile erilaisia muutoksia projektin hahmojen koodeihin ja katso mitä tapahtuu!

Muokkaamisessa luetaan hahmojen koodeja ja valitaan yksi hahmo tai hahmot, joiden koodia halutaan muuttaa. Tapahtumat, jotka käynnistävät hahmojen toiminnan antavat vinkkejä mitkä koodit ovat hyödyllisimpiä. Lopuksi kokeillaan miten erilaiset muutokset koodiin vaikuttavat hahmojen toimintaan. Opettajan on valittava sellainen esimerkkiprojekti, joka tuo hyvin opittavan ohjelmoinnin käsitteen esille. Käsitteen on oltava projektin hahmojen toiminnan kannalta keskeisessä asemassa, jotta oppilaat kiinnittävät siihen huomiota. Lisäksi opettaja voi tukea koodien tulkitsemisessa ja antaa vinkkejä haluttujen muutosten toteuttamiseen.

Vinkki: Computer Science for All sisältää hyviä Scratch-projekteja opiskelijoiden pohdittavaksi hyödyntäen TIPP&SEE -menetelmää. Projektit ovat osa valmiiksi tehtyä opetussuunnitelmaa.

HUOM! Projektien hyödyntäminen vaatii rekisteröitymisen (saat heti sähköpostiin tarvittavat tiedot). Voit kuitenkin ensin katsoa millaisia harjoituksia on tarjolla, ennen kuin rekisteröidyt.

https://www.canonlab.org/scratchact1modules



8.3 5E-MENETELMÄ MATEMATIIKAN JA OHJELMOINNIN YHDISTÄMISEEN

Matematiikka ja tekniikka yhdistyvät ohjelmoinnissa. Ohjelmoinnissa noudatetaan matemaattista täsmällisyyttä ja sillä voidaan toteuttaa mikä tahansa laskutoimitus. Samalla ohjelmointi on myös tietokoneen ohjaamista, jolloin mietitään laitteen teknisten ominaisuuksien mahdollisuuksia. Ohjelmointikielen yleiskäyttöisyys ja teknologiset ominaisuudet piilottavat matematiikan oppilaalta. Matematiikan hyöty ohjelmoinnissa on erikseen opetettava.

Ohjelmoinnissa pyritään aina johonkin tulokseen. Yhdistämällä pyrkimys tulokseen ja mahdollisuus käyttää ohjelmointia laskemiseen, voidaan luoda ohjelmointitehtäviä, jotka harjoittavat matemaattisia taitoja. Jos ei synny tulosta tai se on väärä, on korjattava kirjoitettua koodia. Sopiva ohjelmointiympäristö ja -tehtävä, jotka mahdollistavat virheellisen tuloksen havaitsemisen, tukevat oppimista kokeilemalla. On lisäksi tärkeätä, että ratkaisussa tarvittavat käsitteet nostetaan esille tehtävässä ja tulosten käsittelyssä, jotta aikaansaadaan ymmärtämistä.

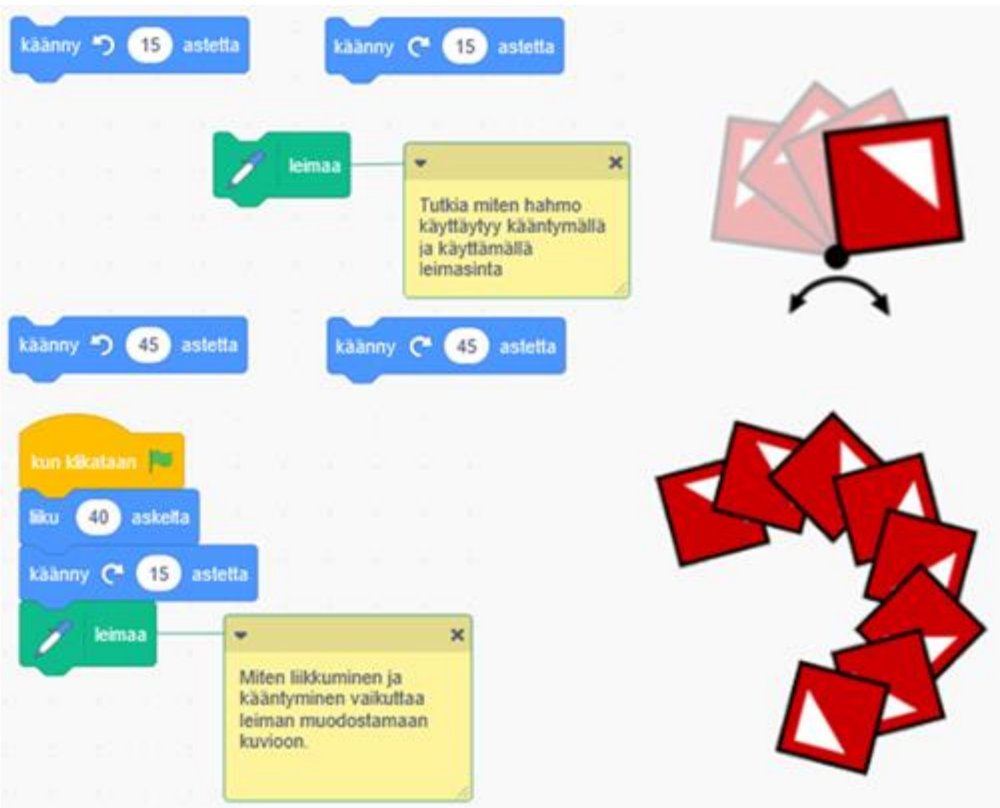
Oppimiseen tarkoitetut ohjelmointikielet ja -ympäristöt ovat kehittyneet merkittävästi. Scratch on tästä hyvä esimerkki: kielen käsitteitä, lohkoja, ei tarvitse muistaa sillä ne voi vetää lohkokirjastosta, lohkon käyttöä helpottaa sen teksti ja miten lohkon muoto sopii toisiin lohkoihin, hahmojen toiminta antaa välitöntä palautetta ja yksittäisen lohkon vaikutusta hahmoon voi myös kokeilla. Helppokäyttöisyys luo kuitenkin jännitteen työkalun ja oppimisen välillä. Koska on mahdollista edetä kokeilemalla ei välttämättä mietitä mitä ollaan tekemässä, vaan tyydytään ensimmäiseksi saatuun tulokseen. Vaihtoehtoisesti saadaan oikea tulos, mutta koodissa on paljon ylimääräisiä lohkoja. Kummassakin tapauksessa koodin merkitystä ei ole ymmärretty ja on luultavaa, että tehtävän tavoite jää myös silloin saavuttamatta.

Edellisen kaltaiseen tilanteeseen voidaan puuttua ohjaamalla tiukemmin koodausta, mutta samalla on varottava ohjelmoinnin muuttumista mekaaniseksi suorittamiseksi. Kyseessä on ohjauksen ja vapauden välinen jännite oppimisen mahdollistamisessa. Oppilaalla on oltava vapaus tutkia ja kokeilla, jotta mahdollistetaan asioiden oivaltaminen. Ohjauksella pyritään varmistamaan, että oppilaat kohtaavat ne asiat, joita on tarkoitus oppia. ScratchMaths-projektissa (https://www.ucl.ac.uk/ioe/research/projects/ucl-scratchmaths) on kehitetty erilaisista tehtävätyypeistä koostuva 5E-menetelmä, jossa kiinnitetään huomiota tasapainoon ohjauksen ja kokeilemisen vapauden välillä. Lisäksi projektissa painotetaan matematiikan oppimisen tavoitetta.

5E-menetelmä:

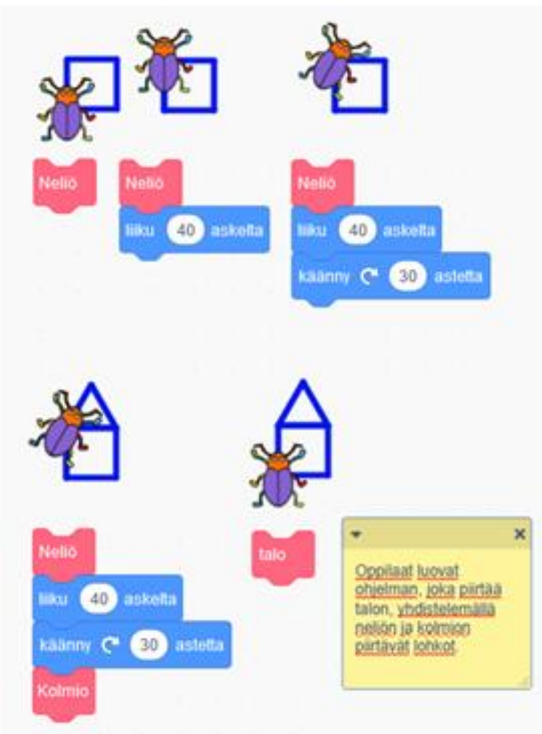
- Explore - Tutkia
- Explain - Kertoa
- Envisage - Kuvitella
- Exchange - Tehdä yhteistyötä
- bridgE - Luoda yhteyksiä

Tutkia-tehtävätyypin tarkoituksena on totuttaa oppilas kokeilemaan ratkaisuja ja oppimaan kokemastaan. Usein oppilaat käyttävät vain niitä ohjelmointikielen käsitteitä, joita he tuntevat ja odottavat että opettaja esittelee heille uudet käsitteet. Itsenäistä otetta voidaan rohkaista tehtävillä, jotka innostavat tutkimiseen, kokeiluun ja virheellisten päätelmien korjaamiseen. ScratchMaths-projektissa tehtävätyyppiä on hyödynnetty Scratchin alkeissa tarjoamalla muutamaa valmista lohkoa hahmon ohjaamiseen ja antaen oppilaan kokeilla niiden vaikutusta. Kun lohkojen toimintaperiaate on ymmärretty, voidaan edetä varsinaisen tehtävän ratkaisemiseen lohkoja hyödyntämällä (ks. Kuva 57).



Kuva 57: Tutkia-tehtävätyyppi, jossa ensin kokeillaan lohkojen vaikutusta hahmoon ja sitten opitun perusteella ratkaistaan monipuolisempi tehtävä.

Kertoa-tehtävätyypissä oppilaan tehtävänä on kertoa, miten hänen koodinsa ratkaisee ongelman ja perustella siinä tekemiään valintoja. Keskeinen osa ratkaisun ymmärtämistä on kyky kertoa siitä. Scratch ohjelmointia aloittaessa oppilaat luovat helposti pitkiä ohjelmia, joiden merkitys on epäselvä. Kun luotua koodia selittää toiselle näkee omat ratkaisut myös eri näkökulmasta. Omien lohkojen luomisen oppiminen helpottaa ohjelman toiminnallisuuden jäsentämistä. Kun lohkoille antaa kuvaavan nimen, tulee koodista luettavampaa (ks. Kuva 58). Silloin on myös helpompi kertoa siitä muille.



Kuva 58: Miten neliö-lohkoon yhdistetään kolmio muodostamaan talo. Kolmio määritellään erilliseksi lohkoksi. Lopuksi yhdistetään neliö ja kolmio -lohkot talo-lohkoksi.

Kuvitella-tehtävätyypissä harjoitellaan koodin suorittamisen seurauksia arvioimalla, miten koodi vaikuttaa hahmon toimintaan. Ohjelmalla tulisi olla tavoite ja oppilaan tulisi pystyä suunnittelemaan, miten tavoite saavutetaan. Kun koodi on valmis voi hän sitten verrata oletusta todellisuuteen. Jos lopputulos ei ole sitä mitä oppilas on ajatellut, voi hän miettiä millaisilla muutoksilla tulos korjattaisiin. Tämä on perusta virheiden korjaukselle nk. debuggaamiselle. Ilman kykyä kuvitella lopputulosta jää tekemisen ja tuloksen välinen yhteys hataraksi. Oppilas etenee silloin satunnaisesti ja ohjelmointi on myös vähemmän mielekästä. Kuvittelua voidaan harjoitella myös ilman tietokonetta, niin että oppilaat ottavat hahmon rooliin ja noudattavat esim. paperilapuille kirjoitettuja komentoja.

Yhteistyö-tehtävätyypissä oppilaat koodaavat ja samalla oppivat yhdessä. Kun kertoo toiselle mitä ajattelee, niin samalla reflektoi omaa osaamistaan. Toinen voi antaa uutta tietoa tai esittää tutun asian uudesta näkökulmasta. Koska kyseessä on alakoulun oppilaat, on huomioitava, että tavoitteellista yhteistyötä vasta harjoitellaan. On tuettava yhteistyön onnistumista ja ristiriitojen ratkaisua. Ohjelmoinnissa on monia asioita mitä voi tehdä yhdessä tai erikseen ja sitten yhdistää tulokset. Suunnittelua ja virheiden korjaamista on luontevaa tehdä yhdessä. Samaa koodia työstettäessä, voi vain yksi käyttää näppäimistöä, mutta jakamalla tehtävä osiin voidaan koodit tehdä erillään ja sitten yhdistää kokonaisuudeksi.

Edellä esitetyt tehtävätyypit voivat edistää matematiikkaa, jos niiden tehtäviin otetaan mukaan laskeminen yhtenä elementtinä. Matematiikan merkitys kannattaa kuitenkin tehdä eksplisiittiseksi. Luoda yhteyksiä -tehtävätyypissä pyritään tuomaan esille tehtävän yhteydet eri aineisiin. Alakoulussa

ohjelmointia opettavat matematiikka ja käsityö, joten ohjelmoinnissa tulisi tehdä yhteys näihin näkyväksi. Ohjelmointi kuuluu myös laaja-alaisen osaamisen taitoihin, joten sen soveltamista tulisi edistää esimerkeillä eri oppiaineista. Jotta oppilaat ymmärtävät yhteyden, on ohjelmointi esitettävä kohteena olevan aiheen kontekstissa tai ulkopuolelta tuotu käsite tulisi esittää myös alkuperäisessä muodossa. On myös hyvä keskustella miten erilaiset kontekstit muuttavat asioita esim. laskeminen ohjelmoimalla ja laskeminen päässä.

8.4 ERILAISIA OHJELMOINNIN TEHTÄVIÄ

Ohjelmaa voidaan tarkastella kahdesta eri näkökulmasta. Ensimmäisessä näkökulmassa tarkastelun tasoilla liikutaan yksittäisistä lohkoista koko ohjelmaan eli tarkasteltavan koodin määrä vaihtelee. Toisessa näkökulmassa ohjelmaa tarkastellaan staattisena kokonaisuutena (ohjelman koodia), dynaamisena kokonaisuutena (ohjelman suoritus) ja kokonaisuutena, jolla on ulkoinen tarkoitus. Näkökulmien yhdistämisellä saadaan aikaiseksi taulukko, jossa on 12 erilaista tarkastelun kohdetta.

Taulukko 7: Ohjelmaa voidaan tarkastella 12 eri tavalla, jotka muodostuvat kahden ulottuvuuden risteymäkohdista.

	Ohjelmakoodi (staattinen)	**Ohjelman suoritus (dynaaminen)**	**Ohjelman tarkoitus ja tavoitteet**
Lohkot	Yksittäiset lohkot	Yksittäisten lohkojen toiminta	Yksittäisen lohkon tarkoitus
Lohko-kokonaisuudet	Syntaktisesti tai semanttisesti (*) yhteen kuuluvat lohkot (eli lohko-kokonaisuudet)	Lohkokokonaisuuksien toiminta	Lohkokokonaisuuksien tavoitteet (alitavoitteet) ja tarkoitus (alitarkoitus)
Lohkokokonaisuuksien suhteet	Lohkokokonaisuuksien väliset suhteet (esim. aliohjelmakutsut)	Lohkokokonaisuuksien suhteiden sarjat (esim. monta aliohjelmakutsua)	Ohjelman tavoitteiden liittyminen alitavoitteisiin ja tarkoituksen alitarkoituksiin
Koko ohjelma	Ohjelmakoodin kokonaisrakenne	Ohjelman sisältämä(t) algoritmi(t)	Ohjelman tavoitteet ja tarkoitus

(*) Syntaksilla tarkoitetaan ohjelmointikielen kielioppia ja semantiikalla merkitystä. Lohkopohjaisissa ohjelmointikielissä syntaktisesti yhteen kuuluvat lohkot ovat esimerkiksi ehtolauseen ja silmukan sisältämät lohkot. Yhteenkuuluvuus semanttisesti vaatii ohjelmakoodin merkityksen ymmärtämistä (joka riippuu ohjelman tarkoituksesta).

Ohjelmointitehtävät voivat liittyä mihin tahansa yhteen näistä kahdestatoista. Seuraavaksi käydäänkin läpi erilaisia tehtäväideoita jokaiseen liittyen.

8.4.1 OHJELMAKOODI

Ohjelmakooditehtävissä tarkastellaan koodia staattisena kokonaisuutena. Koodia ei ole tarkoitus suorittaa tai edes miettiä, miten se silloin toimisi.

8.4.1.1 LOHKOT

Kun tarkastellaan ohjelmakoodia yksittäisten lohkojen tasolla, voi tehtävissä mm. pyytää tunnistamaan tiettyjä lausekkeita tai lauseita (eli tiettyjä lohkoja) tai aliohjelmien ja muuttujien nimiä.

- “Merkitse koodiin matemaattiset lausekkeet.”
- “Merkitse koodiin ehtolauseet.”
- “Luokittele koodin lausekkeet matemaattisiin ja merkkijonolausekkeisiin.”
- “Luokittele koodin palikat lausekkeisiin ja lauseisiin.”
- “Listaa koodin muuttujien nimet.”
- “Ympyröi koodista omien lohkojen nimet.”

8.4.1.2 LOHKO-KOKONAISUUDET

Kun tarkastellaan lohko-kokonaisuuksia, tarkastellaan useampaa lohkoa yhdessä. Tällöin tarkastelun kohteeksi tulevat mm. ehtolauseiden, silmukoiden ja aliohjelmien sisältämät lohkot.

- “Piirrä laatikot ehtolauseen sisältämien lohkojen ympärille.”
- “Mitkä lauseista suoritetaan, kun silmukkaa suoritetaan?”
- “Missä ovat aliohjelman sisältämät lohkot?”
- “Millaisista lohkoista [pitkä ja monimutkainen] lauseke koostuu?”

8.4.1.3 LOHKO-KOKONAISUUKSIEN SUHTEET

Lohko-kokonaisuuksien suhteita kuvaavat esimerkiksi aliohjelmien kutsukohdat. Lisäksi lausekkeiden oikeellisuuden tarkistaminen voidaan laskea suhteisiin, koska oikeellisuus perustuu sen ja sen ympärillä olevien lohkojen suhteisiin.

- “Missä tätä aliohjelmaa käytetään?”
- “Onko tämä lauseke oikein, kun sen pitäisi yhdistää tervehdykseen käyttäjän syöttämä nimi?”

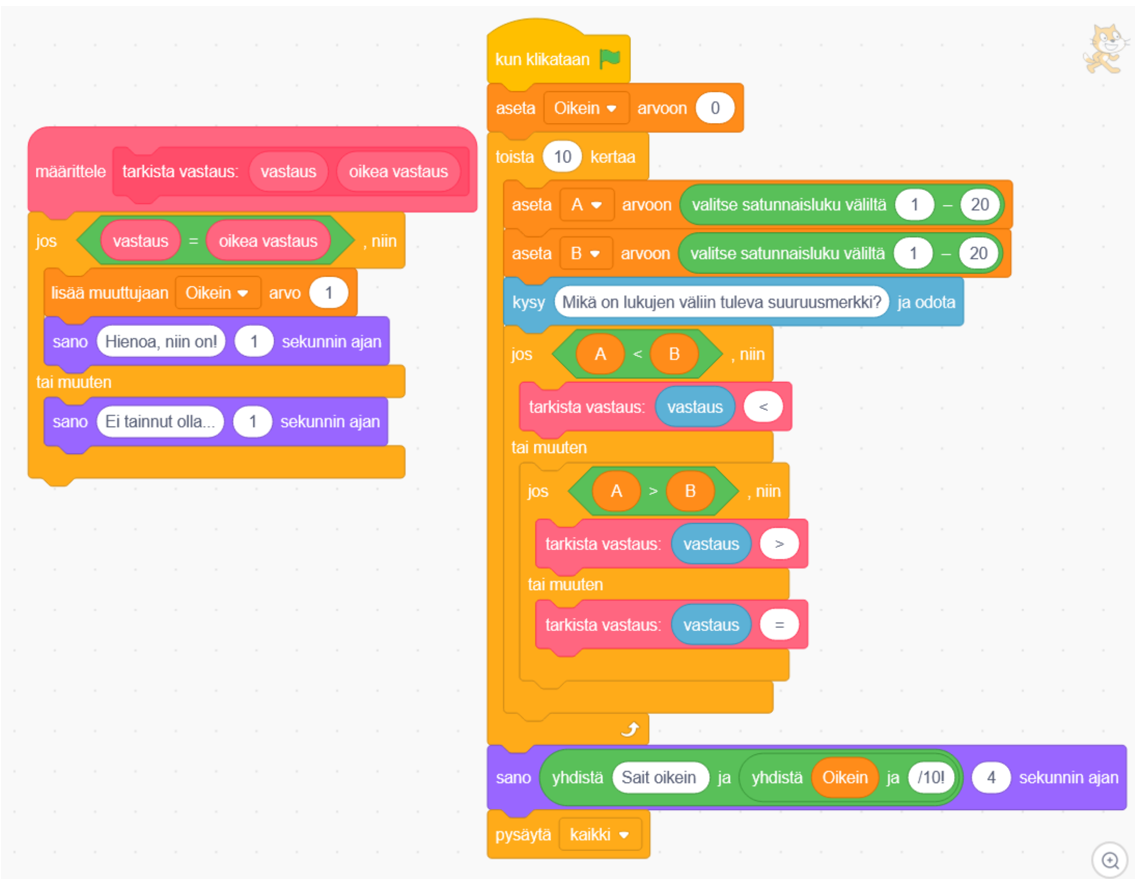
8.4.1.4 OHJELMA

Kun tarkastelussa on koko ohjelma ja ohjelman koodi, perustuu tehtävät usein ohjelmakoodin erilaisiin esitystapoihin ja uudelleen järjestämiseen.

- “Piirrä ohjelmasta prosessikaavio.”
- “Järjestele koodi uudelleen niin, että oikeat koodit ovat oikeilla hahmoilla.”
- “Yhdistä samaan asiaan liittyvät koodit laatikoilla.”

Esimerkki:

Tutkitaan annettua ohjelmakoodia ja vastataan kysymyksiin.



1. Kuinka monta ehtolausetta koodista löytyy? (3)
2. Sisältääkö koodi merkkijonolausekkeen/-lausekkeita? (Ei)
3. Onko seuraavan lausekkeen arvo sama kuin koodin vastaavan lausekkeen? (Kyllä)

yhdistä yhdistä Sait oikein ja Oikein ja /10!

4. Kuvittele vetäväsi nuolet koodissa olevasta aliohjelman määrittelystä jokaiseen kohtaan, jossa aliohjelmaa käytetään. Kuinka monta nuolta vedit? (3)

8.4.2 OHJELMAN SUORITUS

Ohjelman suoritusta tarkkailevissa tehtävissä keskitytään siihen, miten ohjelma toimisi, kun se suoritettaisiin. Oppilaiden on pääteltävä suorituksessa tapahtuvat asiat ohjelman koodista.

8.4.2.1 LOHKOT

Lohkojen suorituksessa huomio kiinnittyy lausekkeiden arvojen päättelemiseen ja lyhyiden lohkosarjojen suorituksen seuraamiseen. Tehtävissä annetaan usein jokin tietty syöte (esim. käyttäjän antama vastaus tai muuttujan arvo), jonka perusteella suoritusta seurataan. Tehtävät eivät sisällä aliohjelmia ja lohkosarjat ovat lyhyitä.

- “Selitä omin sanoin, mitä ohjelmassa tapahtuu, kun käyttäjä syöttää luvun 10.”
- “Mitä jakajat-listassa on ohjelman suorituksen päätyttyä, kun käyttäjä on syöttänyt luvun 10?”
- “Mikä lausekkeen arvo on, kun muuttujan luku arvo on 5?”
- “Mitä ohjelmassa tapahtuu, kun muuttujan luku arvo on 5?”

8.4.2.2 LOHKO-KOKONAISUUDET

Lohko-kokonaisuuksia tarkasteltaessa mukaan tulevat ehtolauseet, silmukat ja aliohjelmat ja lohkosarjojen pituus tai monimutkaisuus kasvavat. Kokonaisuuteen liittyvät lohkot kuitenkin aina liittyvät toisiinsa ja tekevät vain yhden asian.

- “Kuinka monta kertaa silmukka suoritetaan, kun muuttujan piste arvo on 10?”
- “Merkitse koodiin kaikki ne kohdat, joissa kahden muuttujan arvot vaihdetaan.”
- “Järjestele lohkot oikeaan järjestykseen niin, että ohjelma vaihtaa muuttujien A ja B arvot keskenään.”
- “Vaihda toista kunnes … -silmukka toista x kertaa -silmukkaan. Kuinka monta kertaa silmukkaa pitää suorittaa, jotta suoritus vastaa esimerkkiä?”

8.4.2.3 LOHKO-KOKONAISUUKSIEN SUHTEET

Lohkojen ja lohkokokonaisuuksien suhteiden tarkastelussa ei seurata vain tiettyä peräkkäisrakennetta vaan ohjelma sisältää mm. aliohjelmia tai useita peräkkäisiä tai sisäkkäisiä ehtolauseita. Tällöin usein vaaditaan useamman lauseen tai lausekkeen mielessä pitämistä, jotta ohjelman suorituksen seuraaminen onnistuu.

- “Mikä arvo muuttujassa luku on, kun [aliohjelmia sisältävä] ohjelma on suoritettu?”
- “Mitkä arvot muuttuja indeksi [listan indeksi] käy ohjelmassa läpi?”
- “Luettele arvot, joilla sisäkkäisten ehtolauseiden eri haarat saavutetaan.”
- “Millä käyttäjän antamalla vastauksella silmukkaa suoritetaan ikuisesti [kun sitä ei haluta]?”
- “Mikä muuttuja pitää kirjaa listan indeksistä?”

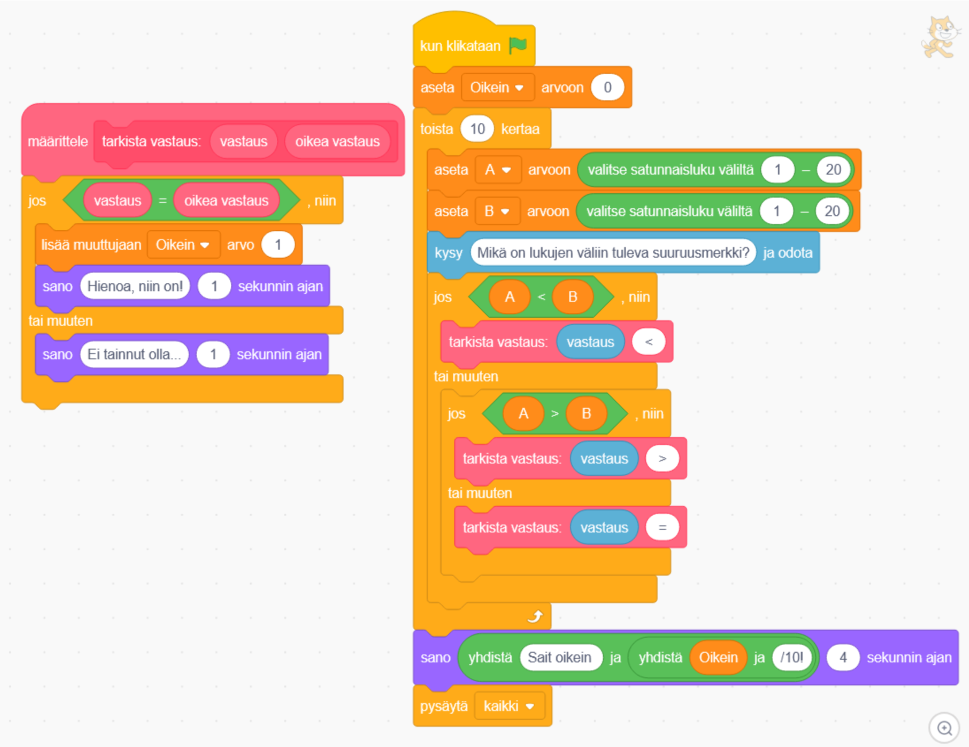
8.4.2.4 OHJELMA

Koko ohjelmaa tarkasteltaessa voidaan mm. tarkistaa, voiko suorituksessa saavuttaa kaikki lohkot tai lohkokokonaisuudet ja arvioida ohjelman suorituksen toimivuutta.

- “Mieti, millä eri tavoilla ohjelman suoritus voi mennä. Onko ohjelmassa lohkoja, joita ei koskaan voida suorittaa?”
- “Mikä seuraavista vaihtoehdoista johtaa esimerkin kaltaiseen lopputulokseen?”
- “Kumpi seuraavista ohjelmista suorittaa annetun tehtävän tehokkaammin?”
- “Anna esimerkki syötteistä, joilla jokainen eri suoritusvaihtoehto saavutetaan.”

Esimerkki:

Tutkitaan annettua ohjelmakoodia ja vastataan kysymyksiin.



1. Mitä ohjelman hahmo sanoo, kun muuttujan A arvo on 3, B arvo 2 ja käyttäjän vastaus <? (Ei tainnut olla…)
2. Kuinka monta vertailua käyttäjältä kysytään? (10)
3. Kuinka monta erilaista testitapausta (eli muuttujien A ja B ja käyttäjän vastauksen yhdistelmää) tarvitaan, että ohjelman jokainen suoritusmahdollisuus voidaan käydä läpi? (6)

8.4.3 OHJELMAN TARKOITUS

Ohjelman tarkoitukseen liittyvissä tehtävissä tarkastellaan ohjelman ja sen osien tavoitteita, tarkoitusta ja hyötyä. Tehtävissä ohjelmaa voidaan tarkastella sekä koodina että suoritettuna ohjelmana.

8.4.3.1 LOHKOT

Yksittäistenkin lohkojen tarkastelussa on tarkasteltava myös ympärillä olevia muita lohkoja ja joissain tapauksessa myös koko ohjelmaa, koska yhden lohkon tarkoitus on sidoksissa koko ohjelman tarkoitukseen. Tehtävien tarkoitus on kuitenkin kiinnittää huomio siihen, että jokaisella lohkolla on jokin tarkoitus ohjelmassa.

- "Mikä seuraavan lausekkeen tarkoitus ohjelmassa on [esim. lauseke, joka muuntaa rahasummia valuutasta toiseen]?"
- "Mikä on ehtolauseen ehdon tarkoitus [esim. tarkistus siitä, ettei käyttäjän syöttämä arvo ole 0, kun sillä on tarkoitus jakaa]?"
- "Mikä seuraavista vaihtoehdoista sopisi ohjelman muuttujan nimeksi?"

8.4.3.2 LOHKO-KOKONAISUUDET

Lohko-kokonaisuuksien tarkastelussa keskitytään kokonaisuuksien tehtävään ja siihen, miten se liittyy koko ohjelman tarkoitukseen.

- "Nimeä uudelleen aliohjelma niin, että se kuvaa aliohjelman tarkoitusta."
- "Mikä on ohjelmaan merkityn osan tavoite?"
- "Merkitse koodiin se kokonaisuus, joka toteuttaa seuraavan tarkoituksen."
- "Selitä, mikä on seuraavan silmukan ja sen sisältämien lohkojen tarkoitus ohjelmassa."

8.4.3.3 LOHKO-KOKONAISUUKSIEN SUHTEET

Lohko-kokonaisuuksien suhteita tarkasteltaessa tarkastellaan mm. aliohjelmia sisältävien kokonaisuuksien tarkoitusta.

- "Valitse vaihtoehdoista sopivin ohjelman muuttujalle muuttuja."
- "Tiivistä [aliohjelmia kutsuvan] lohko-kokonaisuuden tarkoitus yhteen lauseeseen."
- "Järjestele lohko-kokonaisuudet oikeaan järjestykseen niin, että niiden tarkoitukset sopivat peräkkäin."
- "Mitkä seuraavista lohko-kokonaisuuksista toteuttavat saman tarkoituksen?"

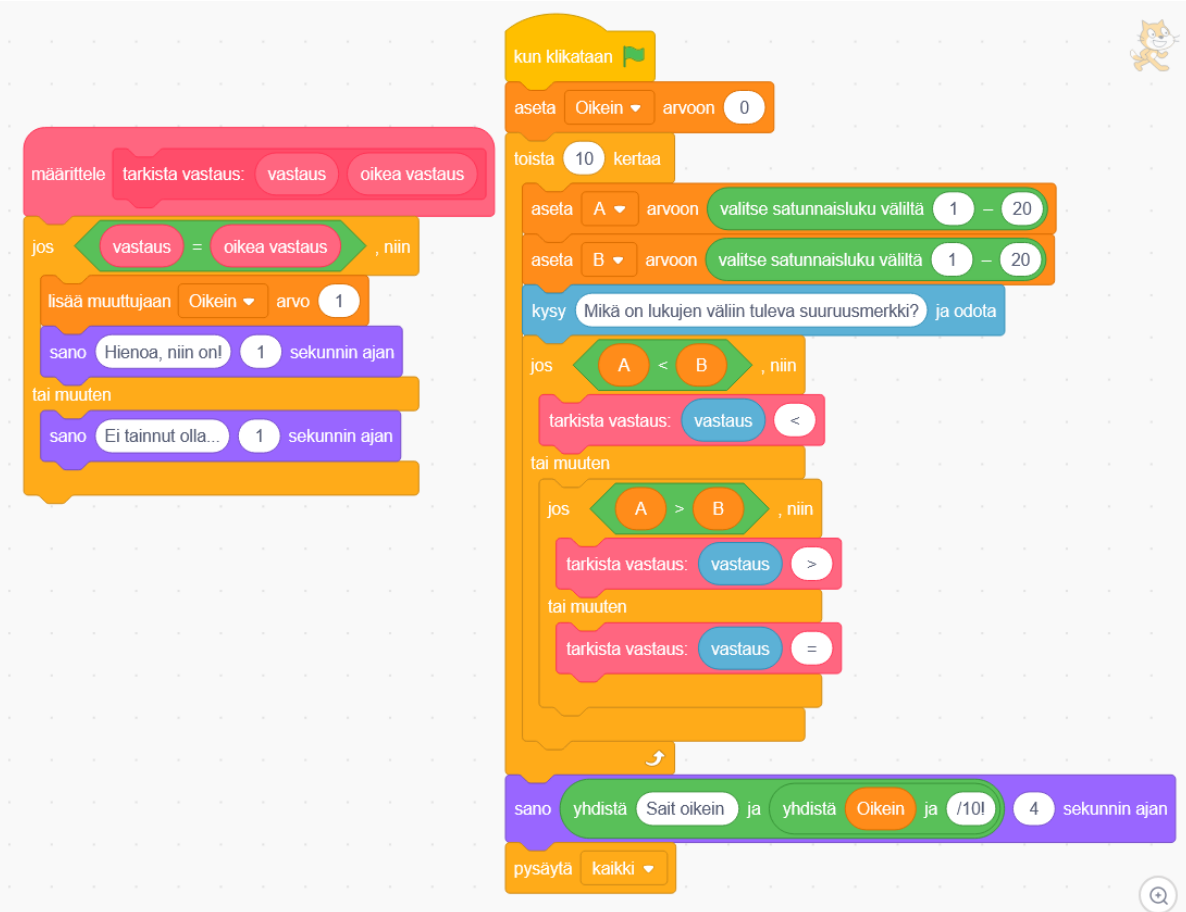
8.4.3.4 OHJELMA

Koko ohjelmaa tarkasteltaessa voidaan ohjelman tarkoitusta ja tavoitteita selittää omin sanoin. Lisäksi koko ohjelman tarkasteluun liittyy myös mahdollisten ongelmatilanteiden selvittäminen ja ohjelman testaus.

- "Nimeä ohjelma."
- "Tiivistä ohjelman tarkoitus yhteen lauseeseen."
- "Valitse seuraavista lauseista se, joka parhaiten tiivistää ohjelman tarkoituksen."
- "Millä eri tavoilla ohjelman toimintaa voisi testata? Mitkä ovat sallitut syötteet ja mitä ohjelman pitäisi tehdä?"

Esimerkki:

Tutkitaan annettua ohjelmakoodia ja vastataan kysymyksiin.



Mikä seuraavista sopii mielestäsi parhaiten ohjelman nimeksi? (kaikki oikein)

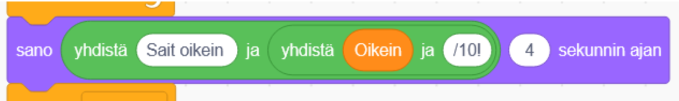
Suuruusvertailuja

Kymmenen suuruusvertailua

Kumpi on suurempi?

Valitse suuruusmerkki!

Mikä on seuraavan lausekkeen tarkoitus? (Muodostaa lause, jonka avulla käyttäjälle voidaan kertoa, kuinka monta vertailua hän sai oikein.)



Mikä seuraavista sopisi parhaiten muuttujan Oikein nimeksi?

oikeita vastauksia [Oikein.]

oikeiden lukumäärä [Oikein.]

vastauksia [Väärin.]

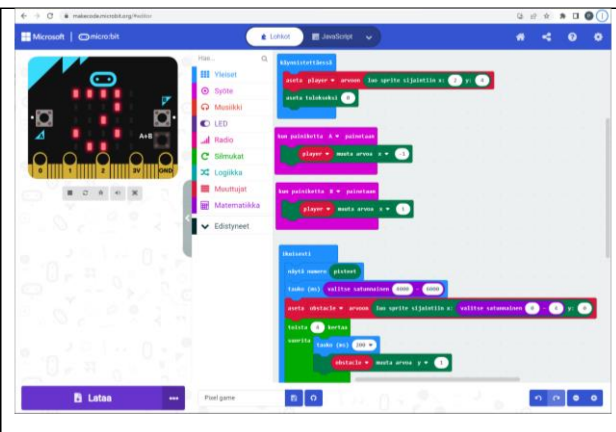
vertailuja [Väärin.]

9 TYÖPAJA 4: OHJELMOINNIN OPETUS

Viimeisessä työpajassa esiteltiin vaihtoehto Scratch-ohjelmointiympäristölle ja hyödynnettiin ohjelmointia matematiikan oppimiseen. Lisäksi käsiteltiin opetuksen suunnittelua. Työpajassa esiteltiin Microsoftin luoma MakeCode, joka on tarkoitettu BBC:n ohjelmoinnin oppimiseen kehittämän korttitietokoneen micro:bitin ohjaamiseen. Korttitietokoneen ohjaaminen, jossa ei ole käyttöjärjestelmää, antaa erilaisen näkökulman tietokonelaitteiston ominaisuuksien hallintaan. Ohjelmointitehtävät, joiden tarkoituksena on tukea matematiikan oppimista, on suunniteltava hyvin. Työpajassa esiteltiin kaksi tapaa: tehtäväaihiot ja mikromaailmat. Tehtäväaihiossa on tarvittava tekniikka valmiina, mutta tehtävän ratkaisemiseksi tarvittava matematiikka on vielä lisättävä tai sitä on muokattava. Mikromaailmassa yksinkertaistettu ympäristö ja sen antama palaute tukee (matemaattisen) tehtävän ratkaisua. Kurssin tavoitteena on ollut tarjota ohjelmoinnin tietoutta opettajille, jotta he osaisivat valita sopivia tehtäviä ja luoda omia ohjelmoinnin opettamiseksi. Työpajassa harjoiteltiin myös ohjelmoinnin opetuksen suunnittelua, joka johdatteli kurssin päättävään ohjelmoinnin oppitunnin suunnitteluprojektiin.

Työpaja alkoi lyhyellä alustuksella, jossa esitettiin yhteenveto edeltävästä verkkosisällöstä ja käytiin lävitse edellisen työpajan kotitehtävien vastaukset. Ensimmäisessä osiossa tutustuttiin Microsoft MakeCode ympäristöön ja korttitietokoneen ohjelmointiin. Toisessa hyödynnettiin ohjelmointia matematiikan oppimiseen. Kolmannessa osiossa harjoiteltiin ohjelmoinnin oppitunnin suunnittelua. Työpajan lopuksi esiteltiin koulutuksen päättävän projektin tehtävänanto.

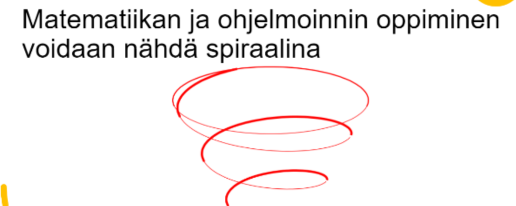
MakeCode ja micro:bit

• Mikä on micro:bit? • micro:bit -korttitietokoneen ohjelmointi MakeCode -editorilla • Esimerkkejä erilaisista micro:bitille tehdyistä ohjelmista • Oman ohjelman luonti	

Ohjelmointi ja matematiikka

• Kokeile -> Muokkaa -> Luo -strategia
 	
• Tavoitteena oppilaan itsenäisen toiminnan tukeminen
 	
• Matematiikan ja ohjelmoinnin vuorovaikutus
 	
• Aikaisemmin opitun hyöty tulee esille oppilaan edetessä tehtävässä
 	
• Osiossa käsitellään kaksi menetelmää
  ○ Tehtäväaihiot
  ○ Mikromaailmat
	Tehtävä 2: Koordinaatiston harjoittelu
	• Tausta: robotilta alkaa virta loppua. Auta häntä pariston luokse.
	• Oppilas ohjaa robottia koordinaateilla lisäämällä/vähentämällä x/y -arvoja.
	• Ensin vihreä lippu, joka alustaa kentän ja laittaa patterin satunnaiseen sijaintiin.

Ohjelmoinnin opettaminen

• Scratch opettajan työkalut
 	
• Ohjelmoinnin opetus
  ○ Opetuksen suunnittelu
  ○ Ohjeistuksen antaminen oppilaille
  ○ Tehtävän tekeminen
  ○ Tehtävän jälkeen
 	
• Ohjelmointitehtävän kehitys
  ○ Mitä ohjelmointitaitoa tehtävä kehittää?
  ○ Miten tehtävä suoritetaan?
  ○ Miten tehtävä esitellään oppilaille?
	Matematiikan ja ohjelmoinnin oppiminen voidaan nähdä spiraalina
	
	Kuva muokattu alkuperäisestä Ag2gaeh https://commons.wikimedia.org/wiki/User:Ag2gaeh CC BY-SA 4.0
	
	Kuva muokattu alkuperäisestä Ag2gaeh https://commons.wikimedia.org/wiki/User:Ag2gaeh CC BY-SA 4.0

Loppuprojekti: ohjelmoinnin oppitunnin suunnitelma

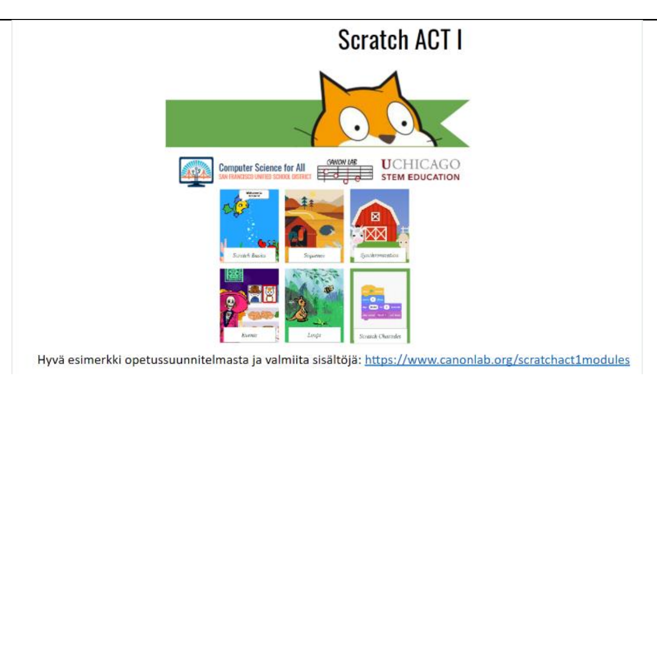
<table>
<tr>
<td>

- Suunnittele kokonainen ohjelmoinnin oppitunti.
    - Mille luokka-asteelle oppitunti on suunniteltu?
    - Mikä on oppitunnin oppimistavoite?
    - Millaisia esitietovaatimuksia oppitunti vaatii?
    - Millaisia materiaaleja oppitunnilla tarvitsee?
    - Mitä oppitunnilla tehdään? (Vaiheittainen kuvaus.)

- Oppitunnin tehtävät voivat olla joko unplugged tai ohjelmointikielellä toteutettavia.

- Kirjoita suunnitelma sellaiseksi, että kollegasi voisi käyttää sitä oppituntina sellaisenaan

</td>
<td>



</td>
</tr>
</table>

10 YHTEENVETO

Edellä oleva sisältö on kooste keväällä 2021 alakoulun luokanopettajille järjestetyn kahden opintopisteen täydennyskoulutuskurssin, ohjelmoinnin opettamisesta matematiikan osana, tekstimateriaaleista. Kohderyhmänä olivat opettajat, jotka kaipasivat opetuksen kehittämiseksi taustatietoa ohjelmoinnista. Lisäksi vastavalmistuneelle opettajalle kurssi tarjosi lähtökohdan ohjelmoinnin opetuksen aloittamiselle. Kurssi antoi viitekehyksen, jolla opettaja voi jäsentää oppikirjojen ohjelmointisisältöjä, täydentää niitä ulkopuolisilla materiaaleilla, luoda yhteyksiä ohjelmoinnin ja matematiikan välille sekä hyödyntää ohjelmoinnin opetusmenetelmiä. Sisältö noudatti alakoulun matematiikan opetussuunnitelmaa ja tarjosi selityksen sen aika lyhyille ohjelmoinnin sisältöjen määritelmille. Ohjelmointi jakautui alkuopetuksen ilman tietokonetta suoritettaviin toimintaohjeisiin ja luokkien 3–6 graafiseen ohjelmointiin. Ohjelmoinnillinen ajattelu määritteli ne taidot, joita harjoittamalla toimintaohjeista on hyötyä myös varsinaisessa ohjelmoinnissa. Graafinen ohjelmointi mahdollistaa kokeilun ja leikillisyyden, mutta samalla voi ohjelmoinnin taidon oppiminen jäädä sivuun. Juuri graafisen ohjelmointiin tarkoitetut opetusmenetelmät mahdollistavat haluttujen taitojen opettamisen ilman, että luovuudesta tarvitse tinkiä. Tekstisisältöjen lisäksi kurssi koostui harjoituksista, työpajoista ja ohjelmoinnin oppitunnin suunnitteluprojektista. Työpajoissa käytännön tekemisen lisäksi keskeistä oli keskustelu, jolloin opettajat saivat kuulla kollegoittensa näkemyksiä ja kokemuksia ohjelmoinnista. Oppituntien suunnittelu päätti kurssin ja se mahdollisti kurssilla opitun soveltamisen käytäntöön.