Kanditutkielma

Tietojenkäsittelytieteen kandiohjelma

Käytettävyysanalyysi SAMPO-ohjelmointikielestä

Théo Friberg

4.4.2022

Matemaattis-luonnontieteellinen tiedekunta

Helsingin yliopisto

Yhteystiedot

PL 68 (Pietari Kalmin katu 5)
00014 Helsingin yliopisto

Sähkopostiosoite: info@cs.helsinki.fi
URL: http://www.cs.helsinki.fi/

HELSINGIN YLIOPISTO – HELSINGFORS UNIVERSITET – UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Koulutusohjelma — Utbildningsprogram — Study programme
Matemaattis-luonnontieteellinen tiedekunta	Tietojenkäsittelytieteen kandiohjelma

Tekijä — Författare — Author

Théo Friberg

Työn nimi — Arbetets titel — Title

Käytettävyysanalyysi SAMPO-ohjelmointikielestä

Ohjaajat — Handledare — Supervisors

yliopistonlehtori Antti Laaksonen ja yliopistonlehtori, dosentti Lea Kutvonen

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
Kanditutkielma	4.4.2022	33 sivua, 10 liitesivua

Tiivistelmä — Referat — Abstract

Tässä työssä tutkitaan 1980-luvulla Oulun yliopistossa kehitettyä Sampo-nimistä ohjelmointiympäristöä, joka yritti tuoda Logo-kielen hyviä ominaisuuksia suomenkielisille peruskoululaisille.

Työ pyrkii vastaamaan kolmeen toisiinsa liittyvään kysymykseen: soveltuuko Sampo opetukseen 2020-luvulla, mitä sen ominaisuuksia voisi tuoda ohjelmoinnin opetukseen yleisellä tasolla ja mitä Sampon kehityksen jälkeen yleistyneitä ominaisuuksia voitaisiin tuoda modernisoituun Sampoon.

Sampon syntaksi ja logiikka nähdään aikaisemmin lausekielisesti ohjelmoineelle suomenkieliselle oppilasryhmälle ymmärrettäviksi. Ryhmän jäsenet päätyivät pienimuotoisessa opetustilanteessa myös omaan kokeiluun.

Samposta ei löydetty yksittäisiä ominaisuuksia, joita voisi tuoda ohjelmoinnin opetukseen yleisellä tasolla. Opetuskokeilun osallistuja kuitenkin nimesi erikoismerkkien suhteellisen puutteen Sampon positiivisena piirteenä ja kielivertailussa huomataan Sampon pieni koko.

Verrokkikielistä taas löydetään joukko Samposta puuttuvia ja varsinkin suuremmissa projekteissa merkitseviä ominaisuuksia. Puutteet liittyvät pääosin pieniin käyttäjä- ja kehittäjäkuntiin sekä vuorovaikutteiseen grafiikkaan ja ohjelmien jäsentämiseen.

Suomenkielinen syntaksi, opetuskokeilun osallistujan tunnistama erikoismerkkien suhteellinen puute ja suuremman projektityön toteuttaminen Sampolla olisivat mahdollisia suuntia jatkotutkimukselle.

ACM Computing Classification System (CCS)
Social and professional topics → Professional topics → History of Computing → History of Programming Languages
Social and professional topics → Professional topics → Computing Education → K-12 Education
Applied computing → Education → Interactive learning environments

Avainsanat — Nyckelord — Keywords

ohjelmointikielet, SAMPO, konnagrafiikka

Säilytyspaikka — Förvaringsställe — Where deposited

Helsingin yliopiston kirjasto

Muita tietoja — övriga uppgifter — Additional information

Sisällys

1 Johdanto

Ohjelmoinnin opetus on riittävän tärkeä osa yleissivistystä, että siitä säädetään perusopetuksen opetussuunnitelman perusteissa (Opetushallitus, 2016). Ohjelmointiin johtavat harjoitukset aloitetaankin jo ensimmäisellä ja toisella vuosiluokalla (Opetushallitus, 2016, s. 129, 235). Tästä huolimatta opetussuunnitelman perusteet eivät ota kantaa siihen, millä kielellä ohjelmointia pitäisi opettaa. Suosittuja valintoja ovat kuitenkin englanninkieliset Python, JavaScript ja Ruby (Liukas ja Mykkänen, 2014, s. 35) sekä suomeksi englannista käännetty graafinen Scratch, jota kirjoittaja opettaa työkseen tiedeluokka Linkissä. Suomalaisissa peruskouluissa on kuitenkin opetettu tietotekniikkaa valinnaisaineena jo 1980-luvun puolivälistä lähtien (Peruskouluasetus 718/1984, 35 §, 36 §). Tältä ajalta on Oulun yliopistossa alun pitäen suomenkieliseksi kehitetty Sampo-ohjelmointiympäristö, jota tässä työssä tutkitaan. Sampo on historiallisesti kiinnostava juuri suomenkielisyytensä takia. Alkuperäisen Sampon kehitys kuitenkin lopetettiin vuonna 1987, eikä sen opetuskäytöstä ole säilynyt dokumentaatiota. Tämän työn anti on pääasiassa historiallinen ja kielisuunnittelullinen: se dokumentoi Sampoa ja sen ominaisuuksia tulevaisuuden kielisuunnittelua ja ohjelmointiopetusta silmällä pitäen.

Kirjoittaja kiinnostui Samposta kuultuaan siitä kanssaopiskelijaltaan Iikka Hauhiolta. Hauhio taas kuuli Samposta Skrolli-lehden toimittaja Ville-Matias Heikkilältä osana vuosina 2017–2018 toteuttamaansa Tampio-projektia. Yhdessä kirjoittaja ja Hauhio toteuttivat Samposta moderneille tietokoneille suunnatun version löydetyn aineiston ja alkuperäisten tekijöiden yhteistyön avulla. Tätä historialliseen uskollisuuteen tähtäävää versiota käytetään tässä työssä kuvatussa opetuskokeilussa.

Työssä tarkastellaan Sampon sopivuutta opetuskäyttöön 2020-luvulla ja sen epäkohtia sekä etuja verrattuna Pythoniin ja Scratchiin. Sopivuutta mitataan opetuskokeilussa kolmella akselilla: onko Sampo ymmärrettävä, ylläpitääkö se oppijoiden kiinnostusta ja saadaanko sillä opetettua jotakin, jota oppijat eivät jo tienneet. Näihin kysymyksiin tutkimus tuottaa puoltavaa aineistoa. Epäkohtia ja etuja taas tarkastellaan pääosin hyödyntämällä kirjallisuudesta poimittuja kriteerejä. Yksi havainto toistui sekä opetuskokeilussa että tässä tarkastelussa: Sampon infiksitoimituksiin liittyvän monimutkaisuuden havaittiin aiheuttaneen sekaannusta myös käytännön kokeessa. Muuten tunnistetut edut ja haitat vaatisivat jatkotutkimusta tullakseen vakuuttavasti osoitetuiksi.

Luvussa 2 käsitellään Sampon historiaa, elvytystyötä ja sen ominaispiirteitä ohjelmointikielenä. Luvussa 3 verrataan Sampoa Scratchiin ja Pythoniin. Luvussa 4 esitellään tätä tutkimusta varten tehdyn lyhyen opetuskokeilun tuloksia. Luvussa 5 tarkastellaan vertailun ja opetuskokeilun tuottamaa tietoa yhdessä ja pohditaan Sampon kehityssuuntia sekä siinä hyväksi todettuja ominaisuuksia. Liitteessä A esitellään opetuskokeilun osana tehty kysely. Liitteessä B esitellään opetuskokeilua varten laadittu oppimateriaali.

2 Sampo-ympäristö

2.1 Historiallista taustaa

Ensimmäisen version Sampo-kielestä toteuttivat Oulun yliopiston tietojenkäsittelyopin laitoksen assistentit Heikki Putkonen, Ilkka Tervonen ja Kari Kuutti vuonna 1983. Tavoitteena oli tuoda suomenkielisten koululaisten saataville Oulun yliopiston professori Pentti Kerolan hiljattaiselta Yhdysvaltain-matkaltaan tuomat innovaatiot: Apple II-koneelle suunnattu Logo-kieli, konnagrafiikka ja MIT:n tutkija Seymour Papertin *Mindstorms*-kirjan opit opetuksesta. (Putkonen, 2021b)

Vuonna 1966 Bolt, Beranek and Newman inc:issä suunniteltiin uusi opetuskäyttöön suunnattu ohjelmointikieli (Solomon et al., 2020, s. 1). Kielen nimeksi tuli Logo ja sen tavoitteena oli antaa kouluikäisille lapsille monipuolisempi kosketus ohjelmointiin kuin he BASICin kaltaisilla kielillä saivat (Solomon et al., 2020, s. 33). Sen takana oli ryhmä, johon kuuluivat muun muassa BBN:n opetusryhmän johtaja Wally Feurzeig, opetusryhmässä työskennellyt Cynthia Solomon ja Papert (Solomon et al., 2020, s. 9, 33). Kielestä tehtiin yhtäältä laajennettava ja toisaalta siihen ohjelmoitiin toteutus konnagrafiikasta (engl. *turtle graphics*). Konnagrafiikan avulla käyttäjä pystyi luomaan graafisia ohjelmia, jossa kuvaa tuotettiin ohjaamalla joko lattialla kulkevaa fyysistä robottia tai sen näytölle piirtävää virtuaalista vastinetta. Konna pyrki ankkuroimaan tietokoneiden oudon maailman intuitiivisesti ymmärrettäviin ilmiöihin kuten lapsen kehon liikkumiseen tilassa. Apple II -tietokoneelle Logoa myymään perustettiin yritys nimeltään Terrapin Software vuonna 1977 (The Logo Foundation, 2015).

Papert julkaisi vuonna 1980 Logon suunnittelussa pohdittuun pedagogiseen teoriaan syventyvän teoksen *Mindstorms: Children, Computers, and Powerful Ideas* (suom. *Lapset, tietokoneet ja ajattelemisen taito*). *Mindstorms* kuvaa lapsen omasta kiinnostuksesta ja tiedonjanosta liikkeelle lähtevän ohjelmoinnin ja matematiikan opettamisen tavan. Itse ohjelmointi on hänen mielestään arvokas oppimiskokemus. Papert kritisoi sekä perinteistä koulumatematiikaa että kirjan kirjoitushetkellä muodissa ollutta uutta matematiikkaa (engl. *New Math*) mielivaltaisesta oppisisältöjen rajauksesta. Hän tarjoaa niiden tilalle konnagrafiikkaa juuri opetuskäyttöön suunniteltuna matematiikan haarana. Teoksen keskiössä on konnagrafiikkaan tutustumisen interaktiivisena välineenä toimiva Logo-kieli ja sen käyttö lasten opetuksen tutkimuksessa. (Papert, 1980)

Sampon lisäksi Mindstormsista ja Logosta kumpusi tunnetumpia projekteja, kuten Lego Mindstorms ja tässä työssä Sampoon vertailtava Scratch. Scratch syntyi MIT Media Laboratoryn alaisessa Lifelong Kindergarten Groupissa, joka ylläpitää sitä edelleen (Resnick et al., 2009, s. 64). Lego Mindstormsiin johtanut robotiikkakehitys alkoi MIT Media Laboratoryssä Papertin johdolla (Martin et al., 2000, s. 2–3). Erilaisia Logon murteita on kehitetty vuosien varrella monia ja Logo Foundation pitää Scratchiä yhtenä niistä (The Logo Foundation, 2015).

Sampoa myytiin Putkosen mukaan lopulta yli kymmeneen kouluun ja siitä tehtiin vuonna 1985 toinen versio (Putkonen, 2021b). Samposta julkaistiin myös muutamia kirjallisia mainintoja: artikkeli vuoden 1984 NordData-konferenssiin (Kuutti et al., 1984), esitelmä Tietokone ja kouluttaja -päiville (Kuutti, 1986), haastatteluartikkelit Insinööriuutisiin (Ylikotila, 1985) ja Mikrolehteen (”Sampo opettaa suomeksi lapset ohjelmoimaan” 1984) sekä lyhyt maininta Dimensio-lehden Logoa käsittelevässä artikkelissa (Kivinen ja Korhonen, 1986).

Kielen ympärille perustettu Systiimi Oy kuitenkin käytännössä lopetettiin vuonna 1987 koulujen palkattoman tietoteknisen tukemisen aiheuttaman lisävaivan takia: yritys myytiin ja sen toimiala, omistajat ja nimi vaihtuivat täysin (Putkonen, 2021b).

Tämän jälkeen Sampo on jäänyt suhteellisen vähälle kirjalliselle huomiolle: ainut kirjoittajan tuntema maininta on tietoteknisen suunnittelun historiallisia umpikujia puinut artikkeli Skrollissa (Heikkilä, 2016). Sen kirjoittajalla ei kuitenkaan ollut muuta kontaktia Sampoon kuin Mikrossa julkaistun artikkelin lukeminen (Heikkilä, 2018, 2021).

Kirjoittaja on yhdessä Iikka Hauhion kanssa selvittänyt Sampon historiaa. Mikro-lehden artikkelista selvisi Heikki Putkosen nimi, ja sen avulla päästiin sähköpostivaihtoon Putkosen kanssa. Putkonen lähetti käskysanalistauksen (Systiimi Oy, 1987b) ja oppimateriaaliksi suunnatun opaskirjan (Systiimi Oy, 1987a). Näitä ei kirjoittajan parhaan tiedon mukaan oltu koskaan julkaistu. Putkosen dokumentteihin kuuluivat myös binäärijakelut kahdesta alkuperäisen Sampon tulkista (molemmat ovat versiota 2.0). Näiden dokumenttien pohjalta kirjoittaja ja Hauhio työstivät uuden toteutuksen Samposta. Työ tapahtui harrastetoimintana ennen tämän tutkielman aloittamista.

Kirjoittajan ja Hauhion työstä innostuneena Putkonen koosti uuden käskysanoja kuvaavan dokumentin (Putkonen, 2020b) ja kehitti ajantasaiset käännösohjeet alkuperäiselle lähdekoodille (Putkonen, 2020a): ohjeita seuraamalla syntyvä emulaattoreille lähtökohtaisesti suunnattu versio sai nimekseen Feenix (Putkonen, 2021b).

Tekstit Systiimi Oy, 1987a; Ylikotila, 1985; Kuutti, 1986; Kuutti et al., 1984 ja yleisemmin materiaalit (kuten binäärit ja käskysanalistaukset), joista tuli tietovarasto Putkonen, 2021a ovat peräisin Putkosen henkilökohtaisista arkistoista. Kirjoittaja on parhaansa mukaan pyrkinyt etsimään käyttökelpoiset viittaukset paikoin leikemäiselle materiaalille. Osa on kuitenkin vaikeasti saavutettavissa. NordData-esityksen tarkka vuosi ei ole selvinnyt aukottomasti kirjoitushetkeen mennessä. Aluksi kirjoittaja ja Hauhio vakuuttuivat siitä, että se esitettiin vuonna 1986 Tukholmassa. Konferenssiyhteenvetoa ei löytynyt kuin ruotsalaisesta kuninkaallisesta kirjastosta. Hauhio tutustui työmatkalla vain lukusalilainaan saatavaan monografiaan, mutta sieltä ei artikkelia löytynyt. Myöhempi sähköpostivaihto Kari Kuutin kanssa täsmentäisi vuodeksi 1984 ja paikaksi pääkaupunkiseudun. Konferenssimuistiinpano kyseiselle vuodelle löytyy eduskunnan kirjastosta, mutta kirjoittaja ei pääse tarkastelemaan sitä tämän työn jättöpäivään mennessä. Vaikeasti saavutettavissa tapauksissa käytettiin Putkosen lähettämiä versioita lähteistä.

2.2 Samposta ohjelmointikielenä

Sampon avainsanat ovat suomeksi ja jopa allatiivin pääte esiintyy muuttujille arvon antamiseen käytettävänä `LLE`-sanana. Kieli on Lisp-sukuisesta esikuvastaan poiketen Forthin sukukieli ja vuonna 1985 julkaistu alkuperäisen Sampon versio 2 toteutettiin Forth 83 -tulkin päälle (Putkonen, 2021c, ”Sampo opettaa suomeksi lapset ohjelmoimaan” 1984). Tämä näkyy käytännössä sanakutsujen suluttomuutena ja riviväleihin liittyvänä rentoutena: sanojen parametrit ja paluuarvot kulkevat koko ohjelmalle yhteisen pinon kautta. Tämä teksti käsittelee pääosin juuri Sampon versiota 2, sillä siitä on säilynyt paremmin dokumentaatiota ja toimivia toteutuksia.

Listaukset 2.1 ja 2.2 näyttävät, miten erilaisilla tavoilla ohjelmakoodia voi jäsentää käyttämällä joko nimettyjä parametreja tai suoraan pinoa. Molemmissa tapauksissa peruslaskutoimitukset puskevat pinon päälle symbolin ja vasta sana `=` jäsentää lausekkeen ja laskee sen. Listauksissa määritellään Fibonaccin lukuja rekursiivisesti laskevat funktiot `FIB` ja `FIB2`. Kommenttimerkkinä toimii Ö. Syy tälle lienee Sampon tapa kuvata ääkkösiä: Forth 83 käyttää kommenttimerkkinä vastakenoviivaa (\), joka kuvautuu Sampon käyttämässä ääkkössubstitutiossa isoksi Ö-kirjaimeksi. Sampot kuitenkin tukevat myös pientä ö-kirjainta kommenttina: sitä esiintyy alkuperäisten Sampojen `SAMPO.BLK`-tiedostoissa (Putkonen, 2021a).

```
LUO :N FIB
    JOS :N < 2 TOSI?
        :N
    MUUTEN
        Ö Välivaiheet joudutaan laskemaan hieman tökerösti
        ( :N - 1 = FIB ) + ( :N - 2 = FIB ) =
    JATKA
VALMIS
```

Listaus 2.1: Esimerkki ohjelman jäsentämisestä nimetyillä parametreilla ja sulkuerotteisilla lausekkeilla. Parametrin `:N` arvo luetaan pinolta implisiittisesti ja sanan suoritus alkaa tilasta, jossa pino on yhtä paikkaa matalampi.

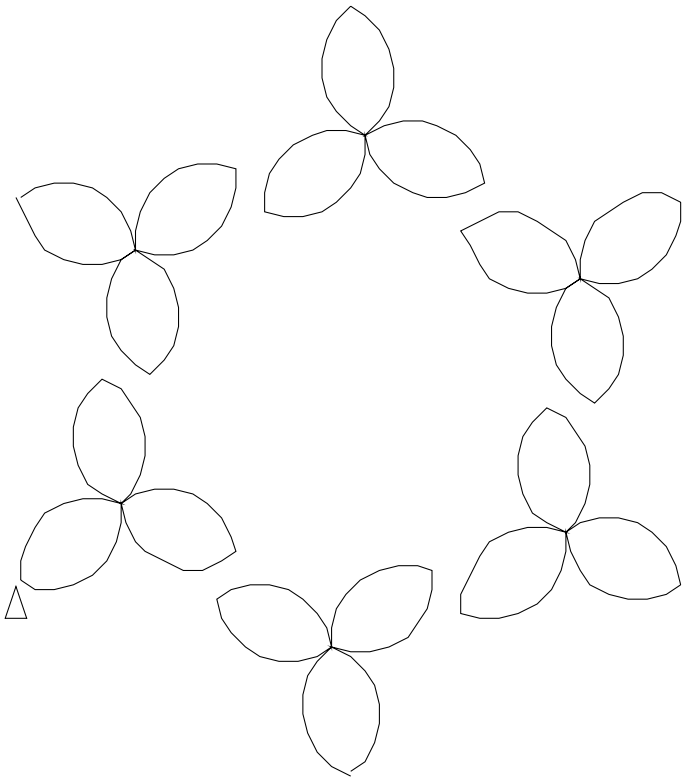
```
LUO FIB2 Ö Kutsuttaessa syö implisiittisesti pinon pään
    TUPLAA Ö Ehdon testaaminen syö arvon pinon päältä
    JOS < 2 VALE? Ö Kun n=0 tai n=1, oikea arvo on valmiiksi pinolla
        TUPLAA Ö Halutaan laskea pinolle n-1 ja n-2
        Ö Sana I puskee pinolle silmukan kierroslaskurin
        2 KERTAA - I = FIB2 VAIHDA VIELÄ?
        + VAIHDA =
    JATKA
VALMIS
```

Listaus 2.2: Esimerkki ohjelman jäsentämisestä ilman nimettyjä parametreja tai sulkuerotteisia lausekkeita. Sana **TUPLAA** kahdentaa pinon päällimmäisen alkion. Sana **VAIHDA** vaihtaa päittäin pinon kaksi päällimmäistä alkiota.

Yksi Sampon päätoiminnallisuuksista on Logosta tuttu konnagrafiikka, jota alkuperäiset Sampot pystyvät piirtämään sekä näytölle että kirjoittimelle. Listaus 2.3 esittelee, miltä yksinkertainen konnagrafiikkaohjelma voi näyttää. Se piirtää kuvan 2.1.

```
Ö SEPPELEEN PIIRTO

LUO :ASKEL KAARI 8 KERTAA :ASKEL ETEEN 15 OIKEA VALMIS
LUO :A APILA
    3 KERTAA :A KAARI 180 OIKEA :A KAARI 60 OIKEA
VALMIS
LUO :A SEPPELE
    6 KERTAA :A APILA YLÖS :A * 20 = ETEEN
    60 OIKEA ALAS
VALMIS

KONNA
100 60 PAIKKA
4 SEPPELE
```

Listaus 2.3: Esimerkki konnagrafiikasta (Systiimi Oy, 1987a, s. 16, ladottu yhdeksi listaukseksi).



Kuva 2.1: Listauksen 2.3 tuottama kuva. Konnan loppusijainti näkyy vasemmassa alakulmassa kolmiona.

3 Sampon, Pythonin ja Scratchin vertailu

3.1 Menetelmistä

Tähän työhön on valittu kaksi eri kirjallista lähdettä, joita vastaan Sampoa ja verrokkikieliä peilataan. Ensimmäinen niistä on artikkeli *Sampo Programming Language* (Kuutti et al., 1984), jossa alkuperäisen Sampon tekijät selittävät Sampon suunnittelutavoitteita ja vertailevat sitä muihin ajan kieliin taulukon 3.2 kriteereillä. Toinen niistä on *A methodology for the analysis of block-based programming languages appropriate for children* (Kraleva et al., 2019), joka vertaa yleisesti lapsille suunnattuja visuaalisen ohjelmoinnin ympäristöjä. Taulukot 3.1, 3.3, 3.4 ja 3.5 perustuvat artikkelista poimittuihin kriteereihin.

Tässä työssä vertaillaan Sampoa, Scratchiä ja Pythonia kahden jälkimmäisen yleisen opetuskäytön takia. Laajasti käytettynä Logon murteena Scratch on myös näyte Sampoa inspiroineen liikkeen nykyaikaisesta tilasta. Kaikki kriteerit ovat alkuperäisteksteissä englanniksi, ja ne ovat tässä vapaasti suomennettuina. Tiedot on selvitetty tarkastelemalla itse ohjelmistoja, niiden dokumentaatioita, verkkosivuja, kirjallisuutta ja Putkoselta saatuja historiallisia tietoja.

Taulukko 3.2 suunniteltiin alkuperäisen Sampon aikoihin vertaamaan ohjelmointikieliä yleisesti. Siitä löytyviä rivejä ”Matriisilaskenta” ja ”Assembler” ei siis pidä tulkita hyvän opetuskielen mittoina. Taulukko 3.2 ei myöskään mittaa myöhemmän tietoteknisen kehityksen varrella vakiintuneita ominaisuuksia. On vaikea sanoa tarkasti, mihin kaikkeen Sampon ajan koulutietotekniikka olisi venynyt. AMC-100-koneessa, jolle Sampoa myytiin, oli sekä värillistä tarkkuusgrafiikkaa että ääniulostuloja (Visiotek, 1985). Värillinen konnagrafiikka sekä yksinkertaiset ääniprojektit olisivat siis voineet olla mahdollisia. Toisaalta Scratchin tapaan yleisemmin multimediaprojekteja, pelejä ja simulaatioita tukeva MicroWorlds-Logo piirto- ja äänityökaluineen julkaistiin vasta 1993 (The Logo Foundation, 2015). Vaikka suoraa teknistä rajoitetta ei olisikaan ollut, Sampo ei ollut tässä muita sukukieliään jäljessä.

Ei ole objektiivisen selvää, minkä ominaisuuden tulkitaan kuuluvan tiettyyn kieleen. Kirjoittaja rajaa tässä kuulumisen kielten perusjakelusta löytyviin valmiisiin ominaisuuksiin,

mutta pyrkii mainitsemaan yleisesti käytetyistä lisäosista löytyvät tai erityisen helposti itse toteutettavat toiminnallisuudet. Pythonin graafisiin käyttöliittymiin suunniteltu `tkinter`-kirjasto on rajapinta ulkoiseen Tk-kirjastoon. Tk ei kuulu kaikkiin Pythonin jakeluihin, mutta rajapinta kuuluu. Tk kuitenkin kuuluu kirjoitushetkellä Python Foundationin Windowsille suosittamaan Python-asennusmenetelmän oletuskokoonpanoon. Samaan kokoonpanoon kuuluu `tkinter`-kirjastolla toteutettu IDLE-ohjelmointiympäristö. Kirjoittaja siis tulkitsee `tkinter`-kirjastosta löytyvät toiminnallisuudet Pythonin ominaisuuksiksi. (Python Software Foundation, 2021)

3.2 Sampon, Pythonin ja Scratchin toteutettamat ominaisuudet

Kielten erilaisia sovelluskohteita vertailevista taulukoon 3.1 valituista käyttötapauksista Sampon käyttöalue on selvästi rajatumpi kuin Scratchin ja Pythonin. Taulukon mittarit on poimittu artikkelista Kraleva et al., 2019. Alla eritellään, millä perusteilla tietyille kielille on merkitty tiettyjen ominaisuuksien tuki.

Ominaisuus	Sampo	Python 3	Scratch
Pelien toteuttaminen	Ei	Kyllä	Kyllä
Animaatioiden toteuttaminen	Kyllä	Kyllä	Kyllä
Valmiudet robotiikka- ja lennokkisovelluksiin	Ei	Ei	Ei
Sisäänrakennetut ohjelmien jakelun toiminnallisuudet	Ei	Kyllä	Kyllä

Taulukko 3.1: Kielten sovelluskohteiden vertailu.

Scratch on suunniteltu interaktiivisiin peleihin (Resnick et al., 2009) ja Python kykenee niihin `tkinter`-kirjastolla ja paketeillaan. Koulurobotiikkaa taas saadaan ohjelmoitua Scratchin laajennuksilla ja esimerkiksi Python-kielisesti EV3DEV-ympäristössä (Mikaiel ja Miao, 2018). Pythonilla on myös mahdollista toteuttaa lennokkiprojekteja (Brand et al., 2018). Scratchiin taas on ainakin kolmannen osapuolen lisäosia, joilla lennokkeja voi hallita (Ishihara, 2018; kebhr, 2021).

Animaatioita voi jossakin muodossa tehdä kaikilla kolmella: Scratchin hahmopohjaisella grafiikalla se on ydinominaisuus (Resnick et al., 2009), Pythonin `tkinter`-kirjastolla ja paketeilla (esimerkiksi `pygame` ja `pyglet`) se onnistuu monipuolisesti ja Sampossa ainakin yksinkertaista viivagrafiikkaa hyödyntävät animaatiot onnistuvat. Sekä Scratch että Python

tukevat värillistä ja interaktiivista konna- ja hahmografiikkaa (Resnick et al., 2009; Scriptics Corporation, ei julkaisupäivää, alaotsikko *Introduction*; Python Software Foundation, 2022, sivu *turtle — Turtle graphics*, värillisyyden suhteen kainaloteksti, interaktiivisuuden suhteen alakohdan *Using events* metodit).

Sekä Python että Scratch sisältävät toiminnallisuuksia ohjelmien jakelemiseen. Pythonin tapauksessa kyseessä on sen mittava keskitetty pakettivarasto PyPI (Python Software Foundation, ei julkaisupäivää[c]). Scratchin tapauksessa tehtävän täyttävät remiksaus- ja jako-ominaisuudet. Sen kehittäjät kokevat yhteisöllisyyden edistämisen keskeiseksi suunnittelutavoitteeksi (Resnick et al., 2009, s. 65–66).

Taulukko 3.2 tarkastelee eri kielten toteuttamia ominaisuuksia. Sen mittarit on poimittu artikkelista Kuutti et al., 1984. Python tarjoaa taulukoon nostetusta jossakin muodossa pitkälti kaiken sen minkä Sampokin. Vastakkainen ei välttämättä pidä paikkaansa: Pythonista pääsee käsiksi assemblerinomaiseen tavukoodiin `dis`-kirjaston kautta ja se tukee kaksiulotteisia taulukoita itsessään ja matriiseja esimerkiksi `numpy`-paketilla. Scratch toteuttaa taulukon ominaisuuksista Sampoa ja Pythonia pienemmän määrän ja niistäkin osan rajatusti: Scratchissä itse määritellyt sanat eivät voi palauttaa arvoa suoraan.[1] Koodiesimerkeistä 3.1, 3.2 ja 3.3 käy kuitenkin ilmi, että kaikilla kolmella kielellä saa toteutettua rekursion.

Ominaisuus	Sampo	Python 3	Scratch
Notaatio	Postfiksi	Infiksi	Infiksi
Koodikoko	Lyhyt	Lyhyt	Palikka
Linkkilistat	Kyllä	Ei	Ei
Tiedostoprimitiivit	Kyllä	Kyllä	Ei
Matriisilaskenta	Ei	Ei	Ei
Laajennettavuus (uusien sanojen määrittely)	Kyllä	Kyllä	Rajattu
Rekursio	Kyllä	Kyllä	Kyllä
Assembler	Ei	Ei	Ei
Pino	Kyllä	Ei	Ei
Grafiikka	Kyllä	Kyllä	Kyllä
Avainsanojen uudelleenmäärittely	Ei	Ei	Ei

Taulukko 3.2: Kielten ominaisuuksien vertailu.

[1] Ominaisuudesta on keskusteltu Scratchin koodiarkiston seikastossa jo viisi vuotta osoitteessa https://github.com/LLK/scratch-vm/issues/79

Alkuperäisen Sampon tekijät nostavat esille kielten käyttämän notaation suunnan: Forthissa kirjoitetaan laskutoimitukset Pythonissa ja Scratchissä käytettävän infiksinotaation $(1 + 2)$ sijaan postfiksinotaatiolla $(1\ 2\ +)$. Tämä johtuu siitä, että parametrit lisätään nimeämisjärjestyksessä pinolle ja kutsuttu sana lukee ne sieltä. Sampo perii postfiksijärjestyksen rajatusti: yleisesti sanojen parametrit annetaan ennen sanaa (esimerkiksi `1 2 PAIKKA` siirtää konnan koordinaatteihin $(1, 2)$), mutta juuri laskutoimitusten tapauksessa kirjoitetaan ensin infiksimuotoinen lause ja kutsutaan sitten `=`-sanaa. Esimerkiksi yhden ja kahden summa lasketaan komennolla `1 + 2 =`. Sampoa käyttävän oppijan täytyykin ymmärtää sekä postfiksijärjestys että missä tilanteissa kuuluukin käyttää tuttua infiksimuotoa.

Rivi ”Koodikoko” ei alkuperäisessä tekstissä ollut tarkemmin määritelty. Vertailussa oli kuitenkin mukana APL-kieli, joka arvioitiin Sampoa lyhytkoodisemmaksi luonnehdinnalla ”minimal”. Sampoa oli kuvattu sanalla ”tight”. (Kuutti et al., 1984, s. 4) APL pyrkiikin lyhyyteen ja formalismiin valitsemalla avainsanoikseen yksimerkkisiä symboleita (Iverson, 1962, s. 345). APL on myös alkuperäisen taulukon ainut matriisilaskentaa tukevaksi mainittu kieli (Kuutti et al., 1984, s. 4). Kirjoittaja ei ole huomannut Pythonin koodikoon olevan yksinkertaisissa esimerkeissä dramaattisesti suurempi tai pienempi kuin Sampon. Scratchin koodikoko on vaikeasti vertailtava sen visuaalisen luonteen takia.

Pythonilla ja Sampolla on tiedostojen käsittelyyn tarvittavat valmiudet, mutta Sampo rajautuu `.BLK`-tiedostoihin. Ne koostuvat yhden kilotavun ruuduista ja matkivat DOS-ympäristössä suorittuvassa Forth-83-tulkissa vanhemmille Forth-järjestelmille tyypillistä tiedostojärjestelmätöntä pääsyä tallennusmediaan: kovalevy yksinkertaisesti jaettiin kilotavun lohkoihin (Perry ja Laxen, 1985). Sampon ruudut jakautuvat vielä tasakokoisiin ja väleillä topattuihin riveihin, eikä Sampo kykene käsittelemään tiedostoja ilman tätä abstraktiota. Scratch ei itsessään toteuta mitään tiedostoprimitiivejä. Siihen on kuitenkin saatavilla Javalla kirjoitetun apuohjelman vaativa laajennos, jonka kautta tiedostoihin pääsee käsiksi (Napier, 2017).

Vain Samposta löytyy tuki linkkilistoille. Niiden lisäksi Samposta löytyy ainoastaan taulukoita. Pythonilla on selvästi vahvin tuki yleisille tietorakenteille. Itse kieleen kuuluu taulukoita, monikkoja ja hakurakenteita. Lisäksi sisäänrakennetusta `collections`-kirjastosta löytyy lisäysjärjestetyksen säilyttäviä hakurakenteita, oletusarvolla varustettuja hakurakenteita, monikkoja nimetyillä kentillä, pakkoja ja monijoukkoja. (Python Software Foundation, 2022, sivu *collections — Container datatypes*) Scratchin muuttujapaletissa taas on tarjolla vain muuttujia ja taulukoita.

Taulukko 3.3 täydentää kuvaa. Taulukon mittarit on poimittu artikkelista Kraleva et al., 2019. Mittareilla tarkastellaan alkuperäisessä yhteydessään visuaalisista ohjelmointikielistä löytyviä tekstikielten ominaisuuksia.

Ominaisuus	Sampo	Python 3	Scratch
Tietotyypit, muuttujatyypit ja vakiot	Kyllä	Kyllä	Kyllä
Peruslaskutoimitukset ja logiikka	Kyllä	Kyllä	Kyllä
Ehtolauseet	Kyllä	Kyllä	Kyllä
Silmukat	Kyllä	Kyllä	Kyllä
Funktiot ja proseduurit	Kyllä	Kyllä	Kyllä
Taulukot ja listat	Kyllä	Kyllä	Kyllä
Osoittimet ja tietorakenteet[1]	Kyllä	Kyllä	Kyllä
Tiedostojen käsittely	Rajatusti	Kyllä	Rajatusti
Koodin jäsentäminen moduuleihin	Rajatusti	Kyllä	Ei
Olio-ohjelmointi	Ei	Kyllä	Ei[2]
Tietokantojen käsittely	Kyllä	Kyllä	Kyllä
Tapahtumarakenteet	Ei	Ei	Kyllä
Piirtotyökalut	Kyllä	Kyllä	Kyllä
Ääni	Ei	Ei	Kyllä
Ohjelmoitaviin hahmoihin perustuva ohjelmarakenne	Ei	Ei	Kyllä
Näppäimistö- ja hiirisyötteen käsittely	Ei	Ei	Kyllä
Tallennusominaisuus	Kyllä	Kyllä	Kyllä

Taulukko 3.3: Kielten ominaisuuksien vertailu.

Perusominaisuuksiksi lukeutuvat kuusi ensimmäistä kohtaa ovat kaikkien kolmen kielen suoraan tukemia. Alla olevat listaukset 3.1, 3.2 ja 3.3 havainnollistavat tätä:

[1] Cons-puut ja linkkilistat. Sinänsä Pythoniin on saatavilla paljon kehittyneempiä tietorakenteita.

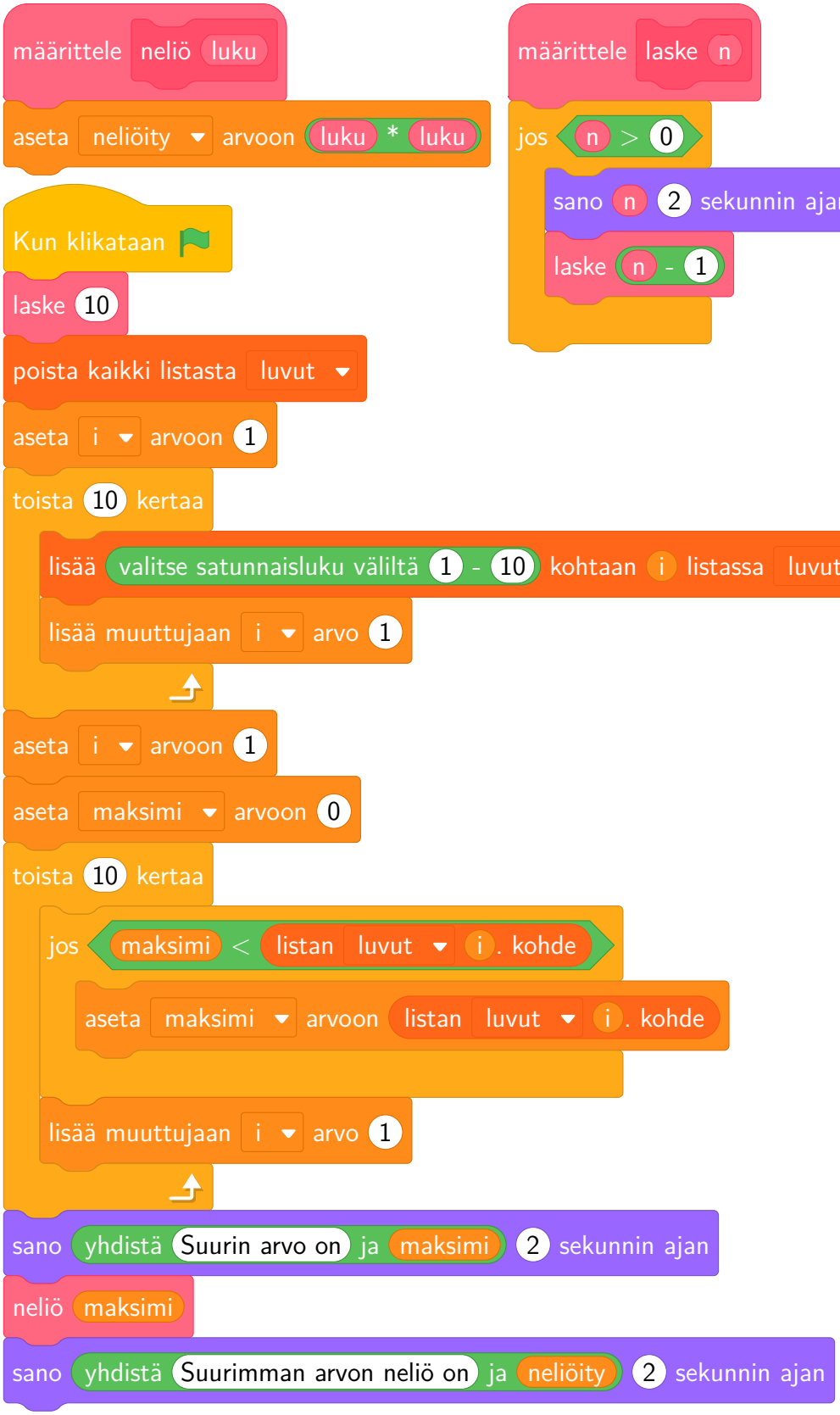
[2] Kralevin, Kralevan ja Kostadinovan mukaan. Kirjoittaja tulkitsee kylläksi.

```
LUO :N LASKE
  JOS :N == 0 TOSI?
    MUUTEN
    :N . UUSIRIVI
    :N - 1 = LASKE
  JATKA
VALMIS

10 LASKE

LUO :x NELIÖ :x * :x = VALMIS

10 JONO LUVUT
10 KERTAA
  1 10 SATTUMA I LUVUT LLE
VIELÄ?
MUUTTUJA MAKSIMI
0 MAKSIMI LLE

10 KERTAA
  JOS I LUVUT > MAKSIMI TOSI?
    I MAKSIMI LLE
  MUUTEN JATKA
VIELÄ?

( SUURIN ARVO ON ) . MAKSIMI ARVO .
UUSIRIVI
( SUURIMMAN ARVON NELIÖ ON ) .
MAKSIMI ARVO NELIÖ .
```

Listaus 3.1: Sampo-kielinen esimerkki tietotyypeistä, muuttujatyypeistä, vakioista, peruslaskutoimituksista, logiikasta, ehtolauseista, silmukoista, funktioista, proseduureista, taulukoista ja listoista. Sampossa proseduurin ja funktion ero on häilyvä pinon kautta implisiittisesti palautuvien arvojen takia: funktio on proseduuri, jonka jäljiltä pinolla on uusia arvoja.

```
from random import randint

def laske(n):
    if n > 0:
        print(n)
        laske(n-1)

laske(10)

def neliö(x):
    return x*x

luvut = [0]*10
for i in range(10):
    luvut[i] = randint(1, 10)

maksimi = 0

for i in range(10):
  if luvut[i] > maksimi:
    maksimi = i

print(f"Suurin arvo on {maksimi}")
print(f"Suurimman arvon neliö on {neliö(maksimi)}")
```

Listaus 3.2: Python-kielinen esimerkki tietotyypeistä, muuttujatyypeistä, vakioista, peruslaskutoimituksista, logiikasta, ehtolauseista, silmukoista, funktioista, taulukoista ja listoista. Pythonissa proseduurit ovat funktioita, joissa ei ole return-avainsanaa.



Listaus 3.3: Scratch-kielinen esimerkki tietotyypeistä, muuttujatyypeistä, vakioista, peruslaskutoimituksista, logiikasta, ehtolauseista, silmukoista, proseduureista, taulukoista ja listoista. Scratchissa itse määritelty lohko ei voi palauttaa arvoa, joten tässä on määritelty funktio proseduurin ja paluuarvomuuttujan yhdistelmänä.

Scratch tukee interaktiivisuutensa pohjana käyttämiään tapahtumarakenteita, syötteen käsittelyä, ääntä sekä hahmoja. Pythoniin nämä ominaisuudet saa paketeilla (esimerkiksi `pygame` ja `pyaudio`), Sampoon ei. Kaikki tukevat jonkinlaista piirtotyökalua sellaisenaan.

Hieman yllättäen kaikki kolme kieltä toteuttavat ainakin rajattuja tietokantaominaisuuksia. Python tukee Sqlite-tietokantaa (Python Software Foundation, 2022, sivu *sqlite3 — DB-API 2.0 interface for SQLite databases*) sellaisenaan ja Sampo tukee omaa `.TKA`-tietokantamuotoaan, joka on erikoistapaus `.BLK`-tiedostosta. Scratch taas tarjoaa pienimuotoisen jaetun tietokannan: kun käyttäjä on saavuttanut tilillään tason ”Scratcher”, hänelle tarjoutuu muuttujaa luodessa vaihtoehto ”Pilvimuuttuja (tallennettu palvelimelle)”. Muuttujat jaetaan suorittuvan ohjelman kaikkien käynnissä olevien esiintymien kesken. Kun ohjelma ei suoritu, ne tallentuvat pysyvästi Scratchin palvelimille. Muuttujia saa olla korkeintaan kymmenen projektia kohti ja ne ovat lukutyyppisiä. (Scratch Foundation, ei julkaisupäivää(c); ceebee, 2019)

Moduulien suhteen on vastaava tilanne: Scratchissä niitä ei voi määritellä, Pythoniin ne kuuluvat kiinteästi ja Sampossa lähin vastine on `.BLK`-tiedostoon talletetun koodin lataaminen. Alkuperäisessä Sampossa on lisäksi `MUISTA`-sana, jolla pystyy tekemään pikalaajennuksen. Pikalaajennustilassa kirjoitettavat komennot peilataan aina auki olevan `SAMPO.BLK`-tiedoston ruudulle 0, josta ne pystyy tallettamaan kovalevylle.

3.3 Sampon, Pythonin ja Scratchin oppimateriaalit

Taulukko 3.4 tarkastelee oppimateriaalin saatavuutta ja laatua. Taulukon mittarit on poimittu artikkelista Kraleva et al., 2019. Alkuperäisessä mittaristossa viimeisellä rivillä mainittu kieli on suomen sijaan bulgaria.

Ominaisuus	Sampo	Python 3	Scratch
Kattava materiaali	Ei	Kyllä	Kyllä
Jäsennelty oppimateriaali	Ei	Kyllä	Kyllä
Videomateriaali	Ei	Kyllä	Kyllä
Tekstimateriaali	Kyllä	Kyllä	Kyllä
Suomenkielinen oppimateriaali	Kyllä	Kyllä	Kyllä

Taulukko 3.4: Oppimateriaalin saatavuutta tutkiva vertailu.

Pythonille ja Scratchille on tarjolla paljon suurempi määrä materiaalia kuin Sampolle ja sitä on tarjolla monipuolisissa muodoissa. Ei kuulu tämän työn piiriin listata kaikkea tarjolla olevaa, mutta taulukon kriteerit on helppo täyttää. Suomeksi löytyy esimerkiksi opettajien kouluttamiseen suunnattu Koodiaapinen, jossa on kappaleittain jäsennetyt kurssit sekä Scratchiä että Pythonia. Siitä löytyy myös videoinserttejä. Koodiaapisen MOOC-osasta ei kuitenkaan ole ollut toteutusta sitten lukuvuoden 2016–2017, ja Scratchin käyttöliittymä on ehtinyt muuttua sen jälkeen (*Koodiaapinen* ei julkaisupäivää). Tiedeluokka Linkki taas tuottaa muun muassa Scratchista ja Pythonista projektimuotoisia materiaaleja, joita voi hyödyntää joko itseopiskelija tai kurssia opettava opettaja. Materiaalikokoelmasta löytyy tekstimuotoisten ohjeiden ohella myös videota (Tiedeluokka Linkki, ei julkaisupäivää).

Alkuperäistä Sampoa varten laaditusta oppimateriaalista on säilynyt vain osa. Kattavin materiaali lienee kirjoittajan ja Iikka Hauhion laatima sanakohtainen dokumentaatio (Hauhio ja Friberg, 2022). Muita kattavia materiaaleja ovat alkuperäisten tekijöiden laatima opaskirja (Systiimi Oy, 1987a) ja käskysanalistaus (Systiimi Oy, 1987b) sekä Putkosen kokoaman Feenix-version dokumentaatio (Putkonen, 2020b). Ohjattuun opetukseen suunnattua materiaalia on tällä hetkellä vain opetuskokeiluun tarvittu oppitunnillinen. Putkonen on lisäksi maininnut alkuperäisen Sampon aikaisia kalvoja, joita kirjoittaja ei ole kuitenkaan nähnyt.

3.4 Sampon, Pythonin ja Scratchin saatavuus ja käytettävyys

Taulukko 3.5 pohtii kielten saatavuutta ja käytettävyyttä. Taulukon mittarit on poimittu artikkelista Kraleva et al., 2019. Se erittelee tavoiteiän, joka nostaa esille kiinnostavan eron kielten välillä: Scratch ja Sampo ovat ohjelmoinnin opetukseen suunnattuja työkaluja toisin kuin Python. Näin niille on ylipäänsä pohdittu kohdeikäryhmät. Scratchin tapauksessa tämä on konkreettinen 8–16-vuotiaat ja sitä vanhemmat (Scratch Foundation, ei julkaisupäivää[a]), Sampossa hieman laveammin peruskoululaiset ja lukiolaiset.

Nimi	Tavoiteikä	Hinta	Kohdealusta	Kielituki
Uusi Sampo	7–18	Ilmainen	Linux ja selain	suomi
Vanha Sampo	7–18	Maksullinen	Aikansa työasemia	suomi
Python	Ei varsinaista	Ilmainen	Monia	englanti
Scratch	8–16	Ilmainen	Selaimet	Monia, mm. suomi

Taulukko 3.5: Ohjelmointikielten saatavuutta tarkasteleva vertailu.

Alkuperäinen Sampo maksoi 600 mk (Putkonen, 2021b) kahta konetta kohden eli nykyrahassa noin 200€ (Tilastokeskus, ei julkaisupäivää). Hintasaraketta tarkasteltaessa on merkitsevää huomata, kuinka tietokoneiden hinnat ovat kehittyneet. IBM PC -klooneja on tuolta aikakaudelta monia, ja juuri Sampon kanssa mahdollisesti käytettyjä ratkaisuja on vaikea selvittää. MikroMikkoja on jo rajatumpi määrä, mutta tarkka Sampon kanssa käytetty malli ei ole tiedossa. Vuonna 1983 grafiikoilla varustettu MikroMikko 1 mallia M4G maksoi vähintään 24 850 mk, joka olisi nykyrahassa noin 9 500€ (Kotilainen, 1983; Tilastokeskus, ei julkaisupäivää). AMC-100-hankintojen hinta pystytään kuitenkin selvittämään yllättävän tarkasti: ”1980-luvun alussa” eli juuri alkuperäisen Sampo-projektin aikoihin pääkoneesta, kuudesta oppilaskoneesta ja muista tarvittavista laitteista ja ohjelmistoista koostuva hankinta maksoi koululle noin 146 000 mk (Saarikoski, 2006). Koneita käytti kerrallaan kahden oppilaan työpari (Visiotek, 1985). Muunnettuna euroiksi hinta olisi ollut vuodesta riippuen vuosina 1980–1985 välillä 74 464,93€ – 49 460,79€ (Tilastokeskus, ei julkaisupäivää).

Kirjoitushetkellä kannettavan tietokoneen, joka tukee Pythonia, Scratchiä ja uutta Sampoa pystyy ostamaan alle kahdellasadalla eurolla. Sampon lisenssi ei ollut yhtä merkittävä menoerä suhteutettuna tietokoneen hankintaan. Uusi Sampo, Python ja Scratch ovat kuitenkin kaikki ilmaiseksi saatavilla (Python Software Foundation, ei julkaisupäivää(a); Scratch Foundation, ei julkaisupäivää(a); Hauhio ja Friberg, 2021).

Kohdealustatkaan eivät ole suoraan vertailukelpoisia: vanhan Sampon aikaan ohjelmointiympäristön käyttö selaimen läpi ei ollut samalla tavalla teknisesti mahdollista. Uusi Sampo, Scratch ja Python ovat kuitenkin kaikki käytettävissä modernissa selaimessa, Python tosin vain Skulptin[1] kaltaisissa epävirallisissa toteutuksissa. Vanhaa Sampoa myytiin Visiotekin AMC-100-koneelle, MikroMikoille ja IBM PC -klooneille (Ylikotila, 1985; Putkonen, 2021a, hakemistot ”Sampo_1”, ”Sampo_2_Imatra” ja ”Sampo_2_Viitasaari”). Uuden Sampon voi asentaa Linux-käyttöjärjestelmälliselle tietokoneelle. Pythonin sivuilta löytyy versiot Windowsille ja MacOSille (Python Software Foundation, 2021). Lisäksi ne viittaavaat ulkopuolisten tahojen ylläpitämiin versioihin Pythonista. Näitä löytyy niin älypuhelimille ja tableteille, kuin palvelimille tai jopa suurtietokoneille suunnatuille järjestelmille (Python Software Foundation, ei julkaisupäivää[b]). Pythonin myös löytää asennettuna monista Linux- ja MacOS-käyttöjärjestelmällisistä tietokoneista valmiiksi (Python Software Foundation, 2022, sivu *Python Setup and Usage*, kohdat ”2.1.1. On Linux” ja ”5. Using Python on a Mac”). Scratchista on tarjolla asennettavat versiot Windowsille, MacOSille, ChromeOSille ja Androidille (Scratch Foundation, ei julkaisupäivää[b]). Linuxille ei löydy

[1] https://skulpt.org/

virallista ladattavaa versiota, eikä se vaikuta olevan tuettu alusta (et451, 2021). Kirjoittaja on kuitenkin saanut asennettua Arch User Repositorystä Windows-asennusohjelman sisällöstä koneellisesti Linuxille uudelleenpaketoidun version.

Kielituessa on vaihtelua: Scratch on tarjolla suomen lisäksi yli seitsemälläkymmenellä kielellä (Scratch Foundation, ei julkaisupäivää[a]) ja Sampo vain suomeksi. Kaikki Pythonin sivuilta saatavat viralliset asennuspaketit ovat englanninkielisiä. Kirjoittaja ei saanut testattua macOS-asennuspakettia, mutta purki sen sisällön ja varmistui siitä, että sen käyttöliittymässä ei ollut tarjolla kielivalintaa (Python Software Foundation, 2021).

3.5 Kirjallisuudesta poimittujen mittarien sivuuttamia eroja

Taulukko 3.6 tarkastelee sellaisia kielten välisiä eroja, jotka artikkelien mittaristot sivuuttivat. Valitut erot nousivat esille tutkielmaa tehdessä ja uutta Sampoa toteuttaessa. Taulukossa on erikseen sarakkeet 1980-luvulla kehitetylle vanhalle Sampolle ja 2020-luvulla kehitetylle uudelle Sampolle.

Ominaisuus	Vanha Sampo	Uusi Sampo	Python 3	Scratch
Merkistökoodaus	7-bittinen ASCII	Unicode	Unicode	Unicode
Muistinhallinta	Käsin	Automaattinen	Automaattinen	Automaattinen
Ohjelmointivirheet johtavat virheviestiin	Usein	Pitkälti aina	Aina	Ei koskaan
Kolmannen osapuolen työkalut	Ei	Ei	Monia	Joitakin

Taulukko 3.6: Eroja, jotka nousivat esille tutkielmaa tehdessä ja uutta Sampoa toteuttaessa.

Kummassakaan Sampon toteutuksessa ei ole merkkijonotyyppiä vaan alkiotyyppi, jota käytetään yleisesti symboleita varten. Esimerkiksi lause `( 1 2 3 )` työntää pinolle listan, jossa on alkiot `1`, `2` ja `3`. Sana **`TEE`**, joka on verrattavissa Pythonin funktioon `eval()` jäsentää ja suorittaa nimenomaan listan alkioita eikä merkkijonoa. Alkuperäinen Sampo käsittelee näitä alkioita 7-bittisellä ASCII-koodauksella, josta on korvattu merkit `] { [ } \ |` merkeillä `Å å Ä ä Ö ö`. Se olettaa myös `.BLK`-tiedostojen olevan tässä muodossa. Uusi Sampo

taas ymmärtää Unicodea ja tulkitsee `.BLK`-tiedostojen rivi- ja ruutujaot merkkimäärän perusteella. Sinänsä näin tulkitsee alkuperäinenkin Sampo, mutta sen merkit ovat aina tavun kokoisia. Uusi Sampo ymmärtää kuitenkin monitavuiset Unicode-merkit kokonaisuuksina. Python 3 käsittee Unicode-merkkejä, tarjoten lisäksi monia muita merkistökoodauksia tarpeen vaatiessa (Python Software Foundation, 2022, sivu *Unicode HOWTO*). Scratch vaikuttaa käyttävän sisäisesti Unicode-merkkejä tukemiensa kielten ja koodivarastosta löytyvän oikealta vasemmalle kirjoitettavien kielten tukeen tähtäävän muutoksen[1] perusteella.

Pythonissa ja Scratchissä on molemmissa automaattinen muistinhallinta. Pythonin tapauksessa se perustuu roskien keruuseen (Python Software Foundation, 2022, sivu *gc — Garbage Collector interface*). Alkuperäisessä Sampossa näin ei ole: listojen tapauksessa käyttäjän kuuluu vapauttaa muisti käsin sanalla `TUHOA`. Uudessa Sampossa `TUHOA` on säilytetty takaisinyhteensopivuussyistä, mutta se vastaa sanaa `POISTA`, joka yksinkertaisesti poistaa pinon päällimmäisen alkion. Yhdistettynä roskien keruuseen tulos on sama kuin alkuperäisessä Sampossa, mutta käyttäjän ei tarvitse hallita muistia käsin.

Python ja molemmat Sampot pyrkivät virhetilanteessa järjestelmällisesti tuottamaan kuvaavan virheviestin. Käytännön kokeilussa alkuperäinen Sampo kuitenkin kaatuu usein antamatta virheviestiä. Alkuperäistä Sampoa on kokeiltu tältä osin DOSBox-ympäristössä ja on mahdollista, että ainakin osa epävakaudesta liittyy emulaatiotekniikkaan. Uusi Sampo tuottaa lähes aina virheviestin, mutta joissakin tapauksissa se saattaa kummuta Scheme-tulkista, jolla se suorittuu. Python taas tuottaa virhetilanteissa poikkeuksia, joihin kuuluu virheviestit (Python Software Foundation, 2022, sivu *Built-in Exceptions*).

Vastapainona Pythonille ja Sampolle Scratch tyypillisesti nielee virheet äänettömästi ja pyrkii tekemään virheelliselläkin syötteellä jotakin mahdollisimman järkevää (Maloney et al., 2010, s. 7). Välillä tämä johtaa yllättäviin tuloksiin: nollalla jakaminen tuottaa laskutoimituksen epäonnistumista kuvaavia (NaN)-arvoja (engl. *Not a Number*), ja muunnoksia absurdinkin oloisissa tyyppiyhdistelmissä tapahtuu implisiittisesti. Esimerkiksi lauseke 

saa arvon (3): ensin 

saa arvon (NaN), sitten se muunnetaan implisiittisesti merkkijonoksi ("NaN") ja lopulta sen pituudeksi lasketaan (3). Sekä Python että Sampo olisivat vastaavissa esimerkissä pysähtyneet nollalla jakamiseen: Python-koodi `len(0/0)` tuottaa **`ZeroDivisionError`**-tyyppisen virheen ja Sampo-koodi `0 / 0 = KPL` tuottaa virheviestin `Virhe: nollalla jako laskussa 0 / 0`.

Kielten välillä on eroja kolmannen osapuolen työkalujen saatavuudessa sekä yleisessä kehitysympäristötuessa. Alkuperäinen Sampo on pikemminkin kehitysympäristö kuin

[1] https://github.com/LLK/scratch-blocks/commit/30a6e67ffa5d5225b88ab3c4745ee7eeede23f00

pelkkä kieli: se sisältää muun muassa oman tekstieditorinsa, johon koodi kirjoitetaan. Uuden Sampon verkkoympäristössäkin on tarjolla värjäävä tekstieditori ja komentorivi. Verrattuna visuaaliseen ja interaktiiviseen Scratch-ympäristöön tai vaikkapa automaattista täydennystä tukevaan IDLEen molemmat jäävät kuitenkin hyvin koruttomiksi.

Pythoniin on monia vaihtoehtoisia kehitysympäristöjä sekä kattavaa joukko paketteja, jotka tekevät monet monimutkaisemmat projektit helpommin toteutettaviksi: esimerkiksi `pygame` ja `pyglet` pelien suunnitteluun, `Pillow` kuvankäsittelyyn, `PyAudio` ääniohjelmointiin ja `pyopencv` konenäköä varten (Python Software Foundation, ei julkaisupäivää[c]). Scratchiin taas on joukko lisäosia, mutta valikoima on selvästi kapeampi. Lisäksi Scratchiin on kolmannen osapuolen työkaluja, kuten Turbowarp-kääntäjä[1]. Sampot voisivat teoriassa olla laajennettavia `.BLK`-tiedostoilla, mutta vaikka `SAMPO.BLK`-tiedostossa on monia kiinnostavia esimerkkejä, niin Sampolle ei ole tarjolla vastaavaa lisäosavalikoimaa.

Yleisesti tähän liittyy kieliä ja ympäristöjä tuottavan yhteisön koko ja tyyppi. Alkuperäistä Sampoa tuotti kolmikko Kuutti, Putkonen ja Tervonen ja uuta Sampoa pari Friberg ja Hauhio. Pythonin ja Scratchin takana on suuret tiimit ja aktiiviset käyttäjäkunnat. Putkonen mainitsee yhdeksi alkuperäisen Sampon kariutumisen syyksi juuri tukitaakan (Putkonen, 2021b).

3.6 Kielivertailun tulokset

Linkkilistojen tuki oli ainut ominaisuus, joka vertailussa löydettiin Samposta muttei verrokkikielistä. Lista Samposta puuttuvista ominaisuuksista, joita vähintään toinen verrokkikielistä tukee on sen sijaan pitkä: sovelluskohteissa interaktiiviset hahmopohjaiset pelit, laajempaa konnagrafiikkaa hyödyntävät projektit, robotit ja lennokit ovat mahdollisia Scratchillä tai Pythonilla. Lisäksi omien tuotosten jakeluun on merkitsevästi huonommat valmiudet Sampolla: tiedostojen ja moduulien tuen rajallisuus vaikuttaa suoraan ohjelmien jäsentämiseen ja remiksaus- sekä pakettitoiminnallisuuksien puute niiden levittämiseen. Oppimateriaalit ja saatavuus ovat kaksi muuta Sampon heikkoa puolta. Oppimateriaaleja on verrokkikielille tarjolla runsaasti ja monipuolisesti. Uuden Sampon saatavuus taas on hieman heikompi verrokkikieliin nähtynä: molemmat muut tukevat vähintään verkkoselaimia ja kahta käytetyintä työpöytäkäyttöjärjestelmää. Uusi Sampo kuitenkin korjaa suurimman osan kehityksensä aikana esiin nousseista ja kirjallisuudesta poimittujen mittaristojen sivuuttamista epäkohdista.

[1] https://turbowarp.org

Ominaisuuksien puutetta voidaan myös tarkastella etuna: Sampo on suhteellisen pieni. Kirjoittajan ja Hauhion sanakohtainen dokumentaatio on kirjoitushetkellä 33 sivua pitkä (Hauhio ja Friberg, 2022) ja se määrittelee kaikki Sampon tunnetut ominaisuudet. Putkosen käskysanalistaus (Putkonen, 2020b) on vielä lyhyempi: 17 sivua. Sampossa on kirjoitushetkellä omia sanoja määrittelemättä 144 käskysanaa ja Scratchissä 126 (Mitattu uudella projektilla, johon on lisätty listamuuttuja listasanojen esiin tuomiseksi muttei mitään laajennoksia. Mukaan on luettu taustan kuusi omaa lohkoa.) Samposta saisi kuitenkin pienehköllä vaivalla karsittua paljon sen ydintoiminnallisuuksiin liittymätöntä vielä pienemmän työkalun luomiseksi: synonyymisanat `K`, `Y`, `A`, `E`, `T`, `O`, `V`, `UR` ja `RIVI`, historiallisista syistä säilytetyt `SIVUSUHDE`, `A:`, `B:`, `KOPIO` ja `TUHOA`, sanojen `TOISTA` ja `JOS` kielteiset muodot sekä sanat `=$`, `#`, `TIETOKANTA` ja `POIMI` poistamalla alitettaisiin Scratchin sanamäärä.

4 Sampo-opetuskokeilu

4.1 Menetelmistä

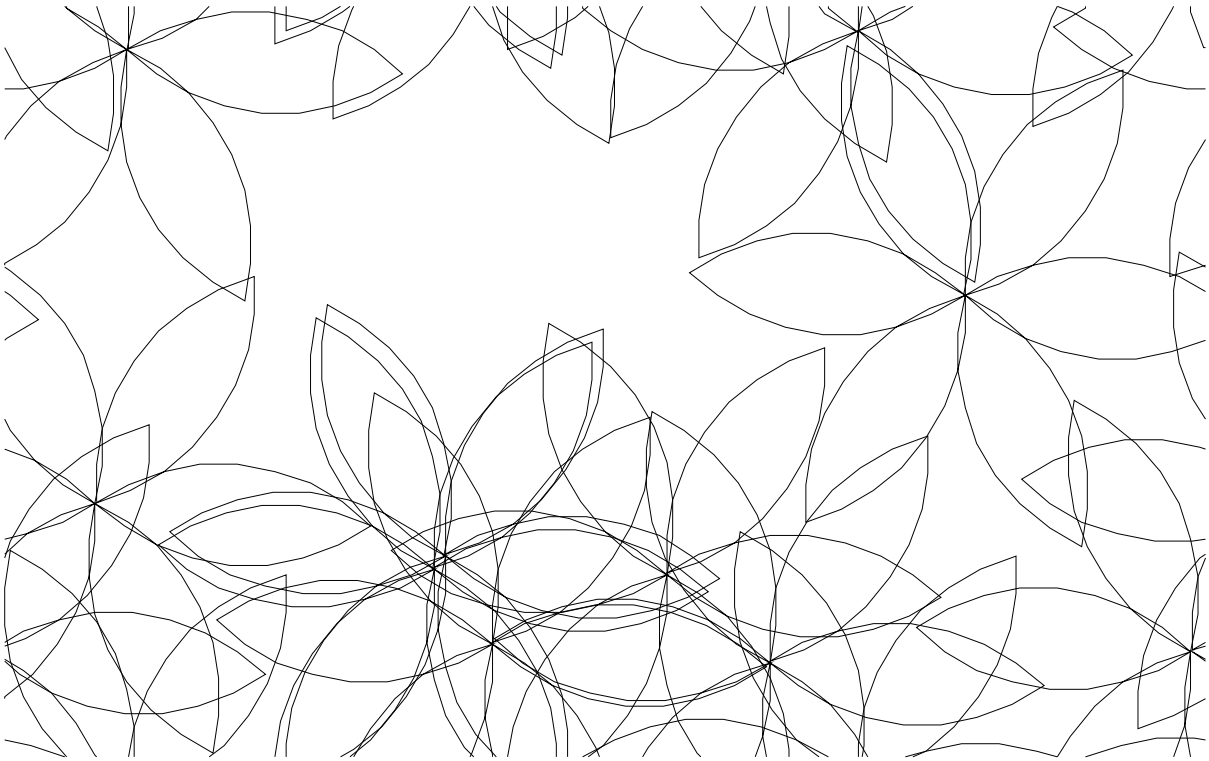
Taulukoihin perustuvan vertailun lisäksi työtä varten toteutettiin opetuskokeilu. Siinä alakouluikäisille vapaaehtoisille ($n = 3$) järjestettiin kahden tunnin mittainen kerhokerta. Osallistujat olivat jo tutustuneet Python-ohjelmoinnin alkeisiin seitsemällä aikaisemmalla kerhokerralla. Kerhokerran päätteeksi osallistujat täyttivät lyhyen kyselyn, joka pyrki mittaamaan heidän ymmärrystään Sampon syntaksista ja konnagrafiikasta sekä mielikuviaan Sampon, Pythonin ja Scratchin vaikeudesta.

Oppitunnilla tarkasteltiin Sampon pinoa, yksinkertaisia laskutoimituksia, omien sanojen määrittelyä **LUO** ... **VALMIS**-rakenteella, `n` **KERTAA** ... **VIELÄ?**-silmukoita ja konnagrafiikkaa sanojen **ETEEN**, **TAAKSE**, **VASEN** sekä **OIKEA** avulla. Oppitunnin loppua kohti toteutettiin yhteisenä projektina listauksen 4.1 kukkaketo. Opetuskokeilussa käytettiin uutta Sampoa, tarkemmin Sampo20-versionhallinnasta muutosta `401ca6e6`. Sampo20-tietovarasto löytyy osoitteesta https://gitlab.com/fergusq/sampo20.

```
KONNA ASEMA NÄYTÄ PINO NÄYTÄ
LUO KAARI
    8 KERTAA 20 ETEEN 10 OIKEA VIELÄ?
VALMIS
LUO TERÄLEHTI
    2 KERTAA KAARI 8 * 10 = VASEN 180 VASEN VIELÄ?
VALMIS
LUO KUKKA
    6 KERTAA TERÄLEHTI 60 OIKEA VIELÄ?
VALMIS
10 KERTAA 640 SATTUMA 400 SATTUMA PAIKKA KUKKA VIELÄ?
```

Listaus 4.1: Oppitunnilla toteutetun kukkakedon ohjelmakoodi.

Opetuskokeilun tulemaan vaikutti monta mahdollista virhetekijää. Osallistujia oli vähän, ja heillä oli aikaisempaa kokemusta tekstiohjelmoinnista. Lisäksi heidät valittiin ohjelmointi-



Kuva 4.1: Oppitunnilla toteutetun kukkaketo-ohjelman (listaus 4.1) ajon tulos.

kerhosta, joten heillä oli selvästi omaa harrastuneisuutta aiheeseen. On siis mahdotonta päätellä, miten edustava peruskouluyleisö, joka ei ole ennen ohjelmoinut tekstikielellä, olisi reagoinut Sampoon. Yleisesti opetuskokeilua voidaan siis pitää suuntaa antavana.

4.2 Opetuskokeilun tulokset

Opetuskokeilun päätteeksi pidetty kysely löytyy liitteestä A. Siinä yksi osallistuja piti Pythonia Sampoa vaikeampana kielenä, muut kaksi pitivät Sampoa kaikkein vaikeimpana. Kaksi osallistujaa oli aikaisemmin ohjelmoinut Scratchillä. Vain yksi osallistuja oli tehnyt konnagrafiikkaa aikaisemmin, ja hän kertoi tehneensä sitä Minecraftilla. Yksi osallistuja vastasi kolmanteen ohjelmointikysymykseen väärin, kaikki muut ohjelmointikysymysten vastaukset olivat oikein. Eräs osallistuja sanoi ensimmäisen ohjelmointikysymyksen vastauksen ääneen, mutta muut eivät tuntuneet reagoineen siihen.

Vapaaseen palautteeseen vastasi aiheeseen liittyen kaksi osallistujaa. Toinen ilmaisi, ettei hänellä ollut palautetta ja toinen piti Sampoa kivempana kuin Pythonia, koska ei tarvinnut käyttää kaksoispisteitä, sulkumerkkejä tai alaviivoja. Kirjoittaja oli tunnin aikana esittänyt Sampon syntaksia Pythonin syntaksiin verraten ja huomauttanut, että esimerkiksi `LUO`

. . . **VALMIS** ja **JOS** . . . **TOSI?** . . . **MUUTEN** . . . **JATKA** -rakenteissa ei käytetä kaksoispisteitä tai sulkuja (joihin kirjoittaja viittasi ”koristeina”) Pythonin tavoin.

Lopputesti osoittaa, että osallistujat ymmärsivät Sampoa hyvin. Tätä päätelmää tukee se, kuinka he oppitunnin aikana tekivät myös omia kokeilujaan: kerhossa syntyi muun muassa VR:n logo, ympyröistä koostuva teos ja kokeilu, jossa näyttö täytettiin kokonaan valkoisella yhdellä suoralla viivalla hyödyntämällä konnapinnan topologisia ominaisuuksia. (Konnapinta on tasoon levitetty torus. Käytännössä tämä johtaa siihen, että oikean laidan yli loikannut konna palaa näytölle vasemmalta laidalta.) Yllättävän vaikeiksi selitettäviksi osoittautuivat kuitenkin pinon kautta kulkevat ja `=`-sanalla auki laskettavat laskut. Kirjoittaja spekuloi, että puhdas infiksi tai postfiksi olisi ollut selkeämpi. Vaikeuskysymys tuottaa hieman ristiriitaista tietoa: Sampoa pidettiin yleisesti vaikeana, vaikkei kuitenkaan yksimielisesti kaikkein vaikeimpana.

Sampo tuntui osallistujien omasta kokeilusta päätellen ylläpitäneen osallistujien kiinnostusta hyvin. Eräs yllätys on, että yksikään osallistuja ei myöskään Sampon hiekkalaatikkomaisesta luonteesta ja vanhahtavasta ilmeestä huolimatta kommentoinut, ettei kyseessä olisi oikeaa ohjelmointia. Linkin ohjelmointikerhojen vetäjät ovat kuvanneet anekdotaalisesti tällaista käytöstä kerhojensa osallistujilta, varsinkin Scratch-kerhon tapauksessa. Kirjoittaja tulkitsee tämän kiinnostuksen merkkinä osallistujien suunnalta.

Kaksi osallistujaa ilmoitti, etteivät he olleet tehneet konnagrafiikkaa ennen. Tarkat vastaukset konnagrafiikkakysymyksiin osoittavat, että Sampolla pystyi oppimaan ainakin konnagrafiikkaan liittyviä taitoja. Python-kerhossa ei oltu vielä käyty **`for`**-silmukoita, mutta niihinkin liittyvät kysymykset ymmärrettiin hyvin. Osallistujilla oli kuitenkin kaikilla aikaisempaa ohjelmointikokemusta, joten tästä seikasta on vähemmän yksioikoista vetää johtopäätöksiä.

Minecraftia aikaisemmin konnagrafiikassa käyttäneen osallistujan vastaukseen liittyen selviää, että `code.org`-sivustolla on Minecraft-aiheinen oppikokonaisuus. Sen visuaalinen ilme ja äänimaailma on kehitetty myötäilemään itse peliä, ja siinä ohjataan visuaalisella ohjelmoinnilla agentiksi nimitettyä konnaan samaistettavaa hahmoa. Agentti ei jätä jälkeä, mutta se kykenee vaikuttamaan Minecraftille ominaisen hilan blockeihin samoilla tavoilla kuin pelaaja kykenisi. Agentin siirtely on myös suhteellisen yksinkertaista: se liikkuu kokonaisia ruutuja ja kääntyy suoria kulmia. Agentin lisäksi käyttäjällä on hallinnassaan Minecraftin hahmoista joko Steve tai Alex. Kappaleiden välillä oppikokonaisuuteen kuuluu videoinserttejä, joissa esiintyy kuuluisia Minecraft-striimaajia.

Kralev, Kraleva ja Kostadinova kuvaavat tekstissään heidän opintokokeiluunsa osallistuneil-

la lapsilla preferenssin kohti `code.org`-sivustoa juuri siellä esiintyvien tuttujen hahmojen takia (Kraleva et al., 2019, s. 7). Siinä valossa on kiinnostavaa, että tämän työn opetuskokeilun osallistujalle muistui aikaisempi konnagrafiikkakokemus Minecraftillä. (Minecraftissä itsessäänkin on agenttihahmo, eli varmuutta juuri `code.org`-kokemuksesta ei ole.)

5 Yhteenveto

Opetuskokeilu näytti Sampon olevan käyttökelpoinen pienimuotoiseen opetuskäyttöön ymmärrettävyyden, kiinnostavuuden ja uuden oppimisen kannalta. Opetuskokeilun osallistujat ymmärsivät Sampon syntaksia ja konnagrafiikan logiikkaa hyvin lopputestin ja osallistujien oman kokeilun määrän perusteella. Oma kokeilu myös tukee johtopäätöstä Sampon kiinnostavuudesta. Opetuskokeilun osallistujat oppivat konnagrafiikan perusteet yhden oppitunnin aikana. Osallistujien käsitys Sampon vaikeudesta oli kuitenkin vaihteleva.

Tutkimuksesta ei selvinnyt yksittäistä Sampon ominaisuutta, jota tuoda ohjelmoinnin opetukseen yleisesti. Suomenkielisen syntaksin vaikutuksia opetuskäyttöön ei tässä tutkimuksessa saatu tarkasteltua syvällisesti: parhain, mitä voidaan varmuudella sanoa on, ettei se estänyt opetuskokeilun onnistumista. Se, kuinka vähän Sampo käyttää erikoismerkkejä ohjelman jäsentämiseen, sai kuitenkin opetuskokeilun osallistujalta kehuja. Sen selvittäminen, vaikuttavatko nämä seikat ohjelmakoodin luettavuuteen, vaatisi lisätutkimusta.

Metriikkatarkastelussa nousi esille joukko suhteellisen pienen mittakaavan ominaisuuksia, jotka voisi olla kiinnostava tuoda Sampoon. Tällaisia ovat esimerkiksi interaktiivinen grafiikka, laajemmat konna- ja hahmografiikkaominaisuudet, tiedostot ja moduulit. Näiden puutteen merkitsevyyden tutkiminen todellisessa opetustilanteessa voisi olla kiinnostava jatkokehityssuunta. Toisaalta ne ovat myös yksinkertaisesti toteutettavissa.

Scratchin remiksaus- ja jako-ominaisuuksista Pythonin pakettiarkistoon molemmista verrokkikielistä paistavat läpi aktiiviset ja suurikokoiset kehittäjä- ja käyttäjäyhteisöt. Näiden työn tuloksena verrokkikielet ovat sovellettavissa paljon laajempaan joukkoon opetustilanteita: jos opettaja haluaa pitää oppitunnin, jossa kehitetään robotiikkasovellus tai toteutetaan interaktiivinen peli, niin siihen löytyy keinot Pythonilla ja Scratchillä. Sampon pieni kehittäjäyhteisö ja olematon käyttäjäkunta ovat vaikeammin parannettavia puutteita. Mahdollisia kehityssuuntia voisi olla joko luoda tiukemmin kohdennettu opetustyökalu rajaamalla Sampo esimerkiksi konnagrafiikan ympärille tai tehdä Samposta yleiskäyttöisempi kehittämällä sille rajapinta johonkin käytetympään kieleen.

Opetuskokeilu ja kirjallisuusanalyyttinen vertailu tuottavat tietoa toisiaan täydentävistä aiheista. Ensimmäinen osoittaa, että yhden oppitunnin mittakaavassa Sampo on vähintäänkin kiinnostava kuriositeetti ja parhaimmillaan omaan kokeiluun herättävä opetustyökalu. Jälkimmäinen valottaa sitä, mihin suuntaan ohjelmointiympäristöt ovat alkuperäisen Sampon

jälkeen kehittyneet ja miten suuri osa nykyisistä opetuskäyttöisen tekstiohjelmointikielen ydinominaisuuksista löytyy jo Samposta. Eri menetelmien tuottamia tuloksia on tässä siis vaikea asettaa vastakkain toistensa pönkittämiseksi tai kumoamiseksi. Yhden oppitunnin aikana metriikkatarkastelussa ilmenevät puuttuvat ominaisuudet tai sovellusalueet eivät osoittautuneet merkitseviksi. Toisaalta jakeluun liittyvä saavutettavuus ja materiaalin saatavuus eivät myöskään tulleet testatuiksi kokeilussa, jota varten uuden Sampon kehittäjät olivat paikalla ja johon oli laadittu erikseen materiaali.

Sampon historiaan liittyy vielä monia avoimia kysymyksiä: kuinka hyvin se toimi alkuperäisessä yhteydessään ja mitä sillä opetettiin? Toisaalta toinen kiinnostava jatkokehityksen suunta on tutkimuksessa havaittujen suunnittelu- ja toteutusepäkohtien paikkaaminen, mahdollisesti uudessa kieliprojektissa joka toisi mukaan Pythonin ja Scratchin hyviä puolia.

Lähteet

Brand, I., Roy, J., Ray, A., Oberlin, J. ja Oberlix, S. (2018). ”PiDrone: An Autonomous Educational Drone Using Raspberry Pi and Python”. Teoksessa: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, s. 1–7.

ceebee (2019). *Updates and Bug Fixes*. URL: https://scratch.mit.edu/discuss/topic/331439/ (viitattu 12. tammikuuta 2022).

et451 (2021). *Can't save project anymore on Linux platform since upgrade to Electron 15*. URL: https://github.com/LLK/scratch-desktop/issues/213.

Hauhio, I. ja Friberg, T. (2021). *Iikka Hauhio / sampo20 · GitLab*. URL: https://gitlab.com/fergusq/sampo20 (viitattu 17. joulukuuta 2021).

– (2022). *Sampo 20*. URL: https://fergusq.gitlab.io/sampo20/documentation/ (viitattu 10. helmikuuta 2022).

Heikkilä, V.-M. (2016). ”Toisenlaiset tietokoneet”. *Skrolli* 1/2016, s. 36.

– (2018, 2021). Henkilökohtainen viestintä. Heikkilä vinkkasi Mikro-lehdestä, mutta ei osannut kertoa muuta.

Ishihara, J. (2018). *scratch2airborne*. Scratch-kirjasto lennokkien lennättämiseen, vaatii vanhan Scratchin version. URL: https://github.com/champierre/scratch2airborne (viitattu 10. tammikuuta 2022).

Iverson, K. E. (1962). ”A Programming Language”. Teoksessa: *Proceedings of the May 1-3, 1962, Spring Joint Computer Conference*. AIEE-IRE '62 (Spring). San Francisco, California: Association for Computing Machinery, s. 345–351. ISBN: 9781450378758.

kebhr (2021). *scratch3-tello*. Scratch-kirjasto lennokkien lennättämiseen, vaatii Scratchin version 3. URL: https://github.com/kebhr/scratch3-tello (viitattu 10. tammikuuta 2022).

Kivinen, A. ja Korhonen, H. (1986). ”Logo-kielen perusteet”. *Dimensio* 3/86.

Koodiaapinen (ei julkaisupäivää). URL: https://koodiaapinen.fi. (viitattu 22. tammikuuta 2022).

Kotilainen, H. (1983). *MIKROMIKKO 1 -myyntikampanja*. URL: https://www.net.fujitsu.fi/fi/historia/mikrotietokone/esitteet/mikromikko1_myyntikampanja1983.pdf (viitattu 17. joulukuuta 2021).

Kraleva, R., Kralev, V. ja Kostadinova, D. (2019). ”A methodology for the analysis of block-based programming languages appropriate for children”. *Journal of Computing Science and Engineering* 13.1, s. 1–10.

Kuutti, K. (1986). ”SAMPO- ja LOGO-kielet tietotekniikan opetuksessa”. Teoksessa: *Tietokone ja kouluttaja -86*. Järjestetty Oulussa 4-6.8.1986. Oulu: Tietomaa. URL: https://raw.githubusercontent.com/sampo-lang/sampo/master/Documenrtaatio/Sampo_DOC_Manuaalikopiot/Tietokone_ja_kouluttaja-86.pdf (viitattu 17. joulukuuta 2021).

Kuutti, K., Putkonen, H. ja Tervonen, I. (1984). ”Sampo programming language”. Vuosiluku Kuutin parhaan käsityksen mukainen. URL: https://raw.githubusercontent.com/sampo-lang/sampo/master/Documenrtaatio/Sampo_DOC_Manuaalikopiot/NordData-86.pdf (viitattu 17. joulukuuta 2021).

Liukas, L. ja Mykkänen, J. (2014). *Koodi2016 – ensiapua ohjelmoinnin opettamiseen peruskoulussa*. 1. painos. Helsinki: Lönnberg Print. ISBN: 978-952-93-4082-8.

Maloney, J., Resnick, M., Rusk, N., Silverman, B. ja Eastmond, E. (marraskuu 2010). ”The Scratch Programming Language and Environment”. *ACM Transactions on Computing Education* 10.4.

Martin, F., Mikhak, B., Resnick, M., Silverman, B. ja Berg, R. (2000). *To Mindstorms and beyond: Evolution of a Construction Kit for Magical Machines*. MIT Media Laboratory.

Mikaiel, A. ja Miao, L. (2018). ”Implementation of a Self-Balancing Robot Using LEGO EV3 Robotic Kit and EV3DEV”. Teoksessa: *Proceedings of the 2nd International Conference on Advances in Image Processing*. ICAIP ’18. Chengdu, China: Association for Computing Machinery, s. 154–158. ISBN: 9781450364607.

Napier, Z. (2017). *File I/O Scratch Extension*. Scratch-kirjasto tiedostojen lukemiseen. URL: https://github.com/Znapi/scratchx-file-io (viitattu 10. tammikuuta 2022).

Opetushallitus (2016). *Perusopetuksen opetussuunnitelman perusteet 2014 — Määräykset ja ohjeet 2014:96*. 4. painos. Helsinki: Next Print Oy, 101, 129, 157, 235, 239, 271, 284, 375, 379 ja 431. ISBN: 978-952-13-5998-9.

Papert, S. (1980). *Mindstorms : children, computers, and powerful ideas*. engl. New York.

Perry, M. ja Laxen, H. (1985). *sampo-lang/sampo*. URL: https://github.com/sampo-lang/sampo/blob/master/Forth_83/Forth_documents/F83%20Users%20Manual.txt (viitattu 5. tammikuuta 2022).

Peruskouluasetus 718/1984 (1984). Annettu Helsingissä 12.10.1984. URL: https://www.finlex.fi/fi/laki/alkup/1984/19840718.

Putkonen, H. (2020a). *Sammon taonta*. URL: https://github.com/sampo-lang/sampo/blob/master/Documenrtaatio/Sammon%20Taonta.odt (viitattu 10. helmikuuta 2022).

– (2020b). *Sampo – Käskysanat*. Feenix-version käskysanat. URL: https://github.com/sampo-lang/sampo/blob/master/Documenrtaatio/Sampo_DOC_Manuaalikopiot/Sampo_k%C3%A4skysanat.odt (viitattu 10. helmikuuta 2022).

– (2021a). *sampo-lang/sampo*. URL: https://github.com/sampo-lang/sampo (viitattu 17. joulukuuta 2021).

– (2021b). *SampoHistoria.md*. URL: https://github.com/sampo-lang/sampo/blob/master/SampoHistoria.md (viitattu 17. joulukuuta 2021).

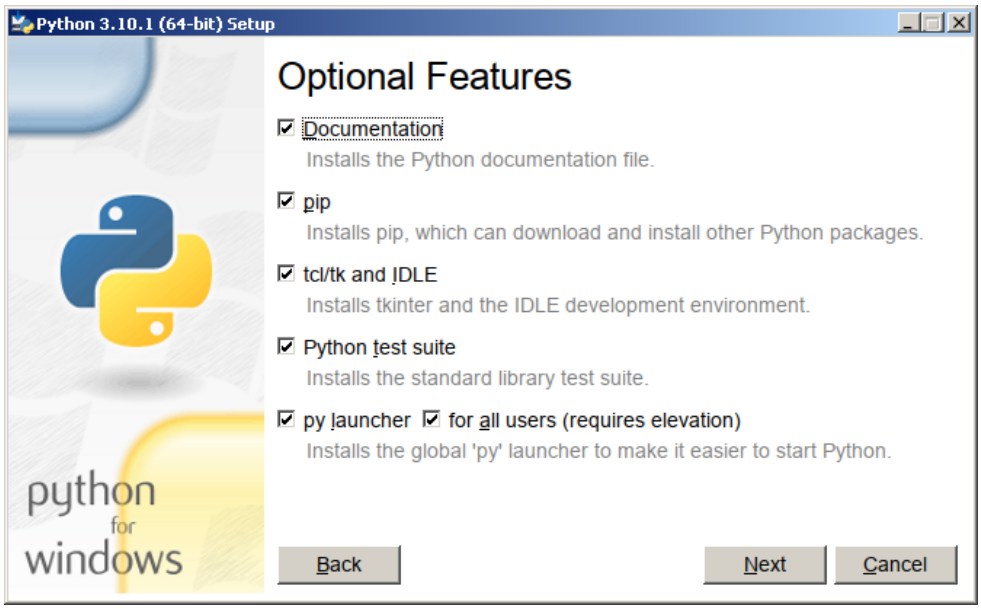
– (2021c). *Säilyneiden Sampo-korppujen versiotiedot*. Korppuja on tietovaraston ylätasolla ja `Documenrtaatio/Sailyneet_Korput/` -kansiossa. DOS-ympäristössä ne tulostavat versiotietoja heti käynnistyessään ja lisää **`CREDO`**-komennolla. URL: https://github.com/sampo-lang/sampo (viitattu 17. joulukuuta 2021).

Python Software Foundation (2021). *Python 3.10.1*. [1] URL: https://www.python.org/downloads/release/python-3101/ (viitattu 11. tammikuuta 2022).

– (2022). *Python 3.10.2 documentation*. URL: https://docs.python.org (viitattu 21. helmikuuta 2022).

– (ei julkaisupäivää[a]). *About Python™| Python.org*. URL: https://www.python.org/about/. (viitattu 17. joulukuuta 2021).

[1] Windowsille suositellun asennusmenetelmän oletusvalinnat:



Python Software Foundation (ei julkaisupäivää[b]). *Download Python for Other Platforms*. URL: https://www.python.org/download/other/. (viitattu 25. tammikuuta 2022).

– (ei julkaisupäivää[c]). *Python Package Index*. URL: https://pypi.org/. (viitattu 17. joulukuuta 2021).

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B. ja Kafai, Y. (marraskuu 2009). "Scratch: Programming for All". *Communications of the ACM* 52.11, s. 60–67.

Saarikoski, P. (2006). "Koneen ja koulun ensikohtaaminen". *Tekniikan Waiheita* 24.3, s. 5–19.

"Sampo opettaa suomeksi lapset ohjelmoimaan" (1984). *Mikro* 4/1984, A–5.

Scratch Foundation (ei julkaisupäivää[a]). *About Scratch*. URL: https://scratch.mit.edu/about/. (viitattu 17. joulukuuta 2021).

– (ei julkaisupäivää[b]). *Download the Scratch App*. URL: https://scratch.mit.edu/download/. (viitattu 21. helmikuuta 2022).

– (ei julkaisupäivää[c]). *Frequently Asked Questions (FAQ)*. URL: https://scratch.mit.edu/faq#clouddata. (viitattu 12. tammikuuta 2022).

Scriptics Corporation (ei julkaisupäivää). *canvas manual page — Tk Built-In Commands*. Pythonin sisäänrakennettu `tkinter`-kirjasto on rajapinta tcl-kieliseen `tk`-kirjastoon. URL: https://tcl.tk/man/tcl8.6/TkCmd/canvas.htm#M14. (viitattu 21. helmikuuta 2022).

Solomon, C., Harvey, B., Kahn, K., Lieberman, H., Miller, M. L., Minsky, M., Papert, A. ja Silverman, B. (kesäkuu 2020). "History of Logo". *Proc. ACM Program. Lang.* 4.HOPL.

Systiimi Oy (1987a). *SAMPO - Opaskirja*. Kirjoittanut todennäköisesti Ilkka Tervonen. Kanervatie 32. 90650 Oulu. URL: https://github.com/sampo-lang/sampo/tree/master/Documenrtaatio/Sampo_DOC_Manuaalikopiot/Opaskirja (viitattu 17. joulukuuta 2021).

– (1987b). *Sampo – Käskysanat*. URL: https://github.com/sampo-lang/sampo/tree/master/Documenrtaatio/Sampo_DOC_Manuaalikopiot/Kaskysanat (viitattu 10. helmikuuta 2022).

The Logo Foundation (2015). *Logo History*. URL: https://el.media.mit.edu/logo-foundation/what_is_logo/history.html (viitattu 17. joulukuuta 2021).

Tiedeluokka Linkki (ei julkaisupäivää). *Materiaalihaku*. URL: https://linkki.cs.helsinki.fi/fi/materials/search. (viitattu 22. tammikuuta 2022).

Tilastokeskus (ei julkaisupäivää). *Rahanarvonmuunnin*. URL: https://www.stat.fi/tup/laskurit/rahanarvonmuunnin.html. (viitattu 17. joulukuuta 2021).

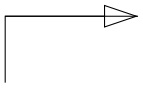
Visiotek (1985). *Multimedianet Teaching System*. vuosiluku videon kuvauksesta. URL: https://www.youtube.com/watch?v=V9wiqFKTly4 (viitattu 17. joulukuuta 2021).

Ylikotila, A. (1985). "Psykologiasta pohja kehitykselle - Oulussa taotaan kouluille SAM-POa". *Insinööriuutiset* 24.4.1985. URL: https://raw.githubusercontent.com/sampo-lang/sampo/master/Documenrtaatio/Sampo_DOC_Manuaalikopiot/Insino%CC%88o%CC%88riuutiset.pdf (viitattu 17. joulukuuta 2021).

Liite A Kyselytutkimus

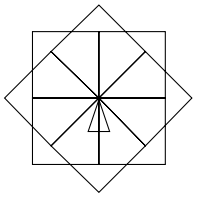
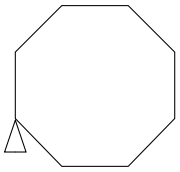
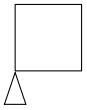
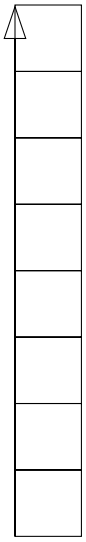
1. Miten vaikeaa SAMPOlla ohjelmointi mielestäsi on?
    1. Vaikeaa
    2. Hieman vaikeaa
    3. Ei vaikeaa eikä helppoa
    4. Hieman helppoa
    5. Helppoa
2. Miten vaikeaa Pythonilla ohjelmointi mielestäsi on?
    1. Vaikeaa
    2. Hieman vaikeaa
    3. Ei vaikeaa eikä helppoa
    4. Hieman helppoa
    5. Helppoa
3. Miten vaikeaa Scratchillä ohjelmointi mielestäsi on?
    1. Vaikeaa
    2. Hieman vaikeaa
    3. Ei vaikeaa eikä helppoa
    4. Hieman helppoa
    5. Helppoa
    6. En ole ohjelmoinut Scratchillä
4. Oletko ennen tehnyt konnagrafiikkaa? Jos kyllä, niin millä?
    1. Kyllä
    2. Ei
5. Minkä kuvista koodi tuottaa?

```
KONNA
100 ETEEN
90 OIKEA
50 ETEEN
```

1.  2.  3.  4. 

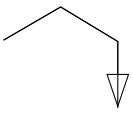
6. Minkä kuvista koodi tuottaa?

```
KONNA
8 KERTAA
4 KERTAA 50 ETEEN 90 OIKEA VIELÄ?
45 OIKEA
VIELÄ?
```

1.  2.  3.  4. 

7. Mikä yksittäinen korjaus saa koodin toimimaan halutulla tavalla eli piirtämään kolmion? Korjauksissa vain alleviivattua osaa on muutettu. Nykyinen koodi tuottaa kuvatun tuloksen.

```
Ö Kolmion piirtäminen TODO: Korjaa

KONNA
3 KERTAA 60 OIKEA 50 ETEEN VIELÄ?
```



1. 6 KERTAA 60 OIKEA 50 ETEEN VIELÄ?
2. 3 KERTAA 120 OIKEA 50 ETEEN VIELÄ?

3. `3` **`KERTAA`** `60` **`VASEN`** `50` **`ETEEN VIELÄ?`**
4. `3` **`KERTAA`** `30` **`OIKEA`** `50` **`ETEEN VIELÄ?`**

8. Onko sinulla palautetta? Sana on vapaa.

   Ehdotuksia: Mikä oli uutta? Mikä oli SAMPOssa kivempaa kuin Pythonissa? Mikä oli Pythonissa kivempaa kuin SAMPOssa? Tuliko ahaa-elämyksiä tai muita kiinnostavia kokemuksia? Jäikö jotakin hampaankoloon?

Liite B Sampo-materiaali opetuskokeiluun

Tässä on opetuskokeilussa käytetty materiaali. Käytännössä se esitettiin hieman sovelletusti ja aikarajoitteen takia sanat `KAARI`, `TERÄLEHTI` ja `KUKKA` jätettiin parametrisoimattomiksi.

B.1 Lyhyt oppimäärä Sampoa Python-kerholaisille

B.2 Vaihe 0: Nettisampo

Avaa selain osoitteeseen https://fergusq.gitlab.io/sampo20/ Aukeavalla sivulla näkyy tämän näköinen laatikko:

Kun pyydetään kirjoittamaan komentoja, ne menevät tähän laatikkoon. Sampo suorittaa kirjoitetun komennon vasta, kun painat enter-painiketta.

B.3 Vaihe 1: Pino, peruslaskutoimitukset ja `.`

Sampo on pinokieli. Tämä tarkoittaa, että muuttujien lisäksi arvoja tallennetaan tietorakenteeseen, joka muistuttaa lautaskasaa: päällimmäiseen lautaseen pääsee käsiksi. Sampon käynnistyessä pino on tyhjä, mutta sen saa alustavasti näkyviin sanaparilla **`PINO NÄYTÄ`**.

Nyt jos pinolle päätyy tietoa, se näkyy tekstikentän yläpuolella. Asetetaan pinolle luku `10`, symboli `+` ja luku 20 kirjoittamalla `10`, `+` ja `20` eri riveille.

Ollaan määritelty yksinkertainen laskutoimitus. Sampon saa laskemaan tuloksen `=`-sanalla: `=` käy läpi pinoa huipulta päin ja laskee sen päältä laskutoimituksen. Lopulta tuloksen saa tulostettua `.`-sanalla komentoriville.

Sampo on yleisesti rento rivivälien suhteen: saman saa kirjoitettua yhdelle riville:

```
10 + 20 = .
```

Pinon käyttö on yleistä Sampossa, mutta erityisen merkittävää on kuinka sanoille annettavat arvot kulkevat pinon kautta. Jos Pythonissa on sana kissa, joka ottaa arvot `a`, `b` ja `c`, niin se kirjoitetaan auki `kissa(a, b, c)`. Sampossa taas se merkitään `A B C KISSA`.

B.4 Vaihe 2: KONNA

Pygletillä piirrettäessä vaaditaan alustuskomentoja. Tyypillisesti nämä ovat Pyglet-kirjaston tuominen ohjelmaan (`import pyglet`), ikkunan luominen

```
ikkuna = pyglet.window.Window(width = 800, height = 600)
```

ikkunan tyhjentäntäminen (`ikkuna.clear()`) ja pelin käynnistäminen (`pyglet.app.run()`). Sitten pygletissä piirtäminen tapahtuu `on_draw`-kutsussa.

SAMPOssa on vähemmän käynnistettävää: piirtoalue tyhjennetään sanalla `KONNA`. Sampossa nimittäin piirretään siirtelemällä valkoisen jäljen jättävää konnaa. Piirtoalueen tyhjentämisen lisäksi sana `KONNA` asettaa konnan aloitussijaintiin.

Konnan sijainnin ja kulman saa näkyviin sanaparilla `ASEMA NÄYTÄ`.

B.5 Vaihe 3: Leikitään: ETEEN, TAAKSE, VASEN, OIKEA

Konnaa ohjataan sanoilla `ETEEN`, `TAAKSE`, `VASEN` ja `OIKEA`. `ETEEN` ja `TAAKSE` siirtävät konnaa tietyn askelmäärän ja `OIKEA` ja `VASEN` kääntävät sitä tietyn astemäärän (joita kokonaisessa kierroksessa on 360)

Esimerkiksi näin saa piirrettyä neliön

```
100 ETEEN 90 OIKEA
100 ETEEN 90 OIKEA
100 ETEEN 90 OIKEA
100 ETEEN 90 OIKEA
```

B.6 Vaihe 4: KERTAA ja LUO

Tämä on kuitenkin suhteellisen vaivalloista: toistuvat vaiheet voi lyhentää `KERTAA ... VIELÄ?`-sanaparilla:

```
4 KERTAA 100 ETEEN 90 OIKEA VIELÄ?
```

Lisäksi Pythonista tuttua **def** -avainsanaa vastaa Sampossa sanapari **LUO ... VALMIS**. Voimme siis määritellä oman sanan neliölle

```
LUO NELIÖ 4 KERTAA 100 ETEEN 90 OIKEA VIELÄ? VALMIS
```

Nyt voimme käyttää tätä sanaa monimutkaisemmassa ohjelmassa, esimerkiksi:

```
PINO NÄYTÄ
KONNA
ASEMA NÄYTÄ

LUO NELIÖ
    4 KERTAA 100 ETEEN 90 OIKEA
VALMIS

6 KERTAA NELIÖ 30 ETEEN 60 OIKEA VIELÄ?
```

Pidemmät ohjelmat kannattaa kirjoittaa `suorita!`-napin alapuolella olevaan tekstikenttään ja suorittaa `suorita!`-napilla. Pidempää ohjelmaa saa siten korjailtua ja ajeltua uudelleen. (Lisäksi jos Sampon saa ikuiseen silmukkaan tai välilehden vahingossa sulkee, koodi tallentuu)

`LUO`-sanalla luotuun sanaan voi lisäksi määritellä parametreja:

```
LUO :A :B MIINUS
    :A - :B =
VALMIS
```

B.7 SAMPO-projekti 1: Kukkaketo

Lähdetään hahmottelemaan yksinkertaista kukkaa. Terälehdet muodostuvat kahdesta ympyrän kaaresta ja kohtaavat kukan keskellä. Aloitetaan alustamalla konna, pinon esitys ja konnan sijainnin näyttäminen

```
KONNA ASEMA NÄYTÄ PINO NÄYTÄ
```

Kaaren saamme Sampolla aikaan **KERTAA**-silmukalla näin:

```
8 KERTAA 10 ETEEN 10 OIKEA VIELÄ?
```

Luvut 8, 10 ja 10 eivät ole tässä mitään erityisen tärkeitä, niistä sattuu syntymään kivan näköinen kaari. Eteen kuljettua matkaa vaihtelemalla saadaan skaalattua kaarta. Tämän avulla voidaan määritellä sana KAARI

```
LUO :KOKO KAARI
    8 KERTAA :KOKO ETEEN 10 OIKEA VIELÄ?
VALMIS
```

Terälehden saamme aikaan yhdistämällä kaksi kaarta:

Jos kokeilemme

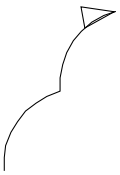
```
10 KAARI 10 KAARI
```

Saammekin tällaisen tuloksen:

Kaarten välissä pitää ilmeisesti kääntyä. Kokeillaan kääntyä kaarta piirtäessä käännytty määrä toiseen suuntaan (piirrettäessä kaarta käännytään kahdeksan kertaa kymmenen astetta):

```
10 KAARI 8 * 10 = VASEN 10 KAARI
```

Vieläkään ei käännytty tarpeeksi



Käännytään vielä ympäri ennen toisen kaaren piirtämistä

```
10 KAARI 8 * 10 = VASEN 180 VASEN 10 KAARI
```

Näyttää jo selvästi paremmalta, mutta konna osoittaa lopuksi kummaan suuntaan. Käännytään toisen kaaren aikana käännytty matka takaisin vasemmalle ja käännytään sitten toisen kerran ympäri.

```
10 KAARI 8 * 10 = VASEN 180 VASEN
10 KAARI 8 * 10 = VASEN 180 VASEN
```

Tätä voi lyhentää: koodi toistuu kahdesti, joten sen voi ilmaista kertaa-silmukalla. Nyt kun olemme saaneet määriteltyä terälehden, tehdään siitäkin oma sanansa.

```
LUO :KOKO TERÄLEHTI
    2 KERTAA :KOKO KAARI 8 * 10 = VASEN 180 VASEN VIELÄ?
VALMIS
```

Nyt voimme piirtää kokonaisen kukan. Tätä varten piirrämme terälehtiä ja käännymme niiden välissä, kunnes päädymme takaisin aloituskulmaan. Koodimme siis on jotakin tämän muotoista:

```
??? KERTAA 20 TERÄLEHTI ??? VASEN VIELÄ?
```

Valitaan tässä luvut 9 ja 60. Mikä tahansa lukupari kelpaa, kunhan niiden tulo on 360: tällöin käännytään kokonainen ympyrä. Määritellään tästäkin oma sanansa:

```
LUO :KOKO KUKKA
    6 KERTAA :KOKO TERÄLEHTI 60 OIKEA VIELÄ?
VALMIS
```

Nyt osaamme piirtää erikokoisia kukkia. Tehdään nyt silmukka, joka piirtää niitä kokonaisen kedon.

```
10 KERTAA 20 KUKKA VIELÄ?
```

Paitsi että kukat päätyvät kaikki samaan paikkaan. Tämän välttämiseksi voimme käyttää komentoa **SATTUMA**, joka arpoo satunnaisluvun nollan ja sille annetun luvun väliltä. (Hyödyksi on myös tieto, että piirtoalueemme vaakamitta on 640 kuvapistettä ja pystymitta 400 kuvapistettä) Toinen hyödyllinen komento on **PAIKKA**, joka siirtää konnan koordinaatteina annettuun sijaintiin. Voimme siis justeerata silmukkaa näin:

```
10 KERTAA 640 SATTUMA 400 SATTUMA PAIKKA 20 KUKKA VIELÄ?
```

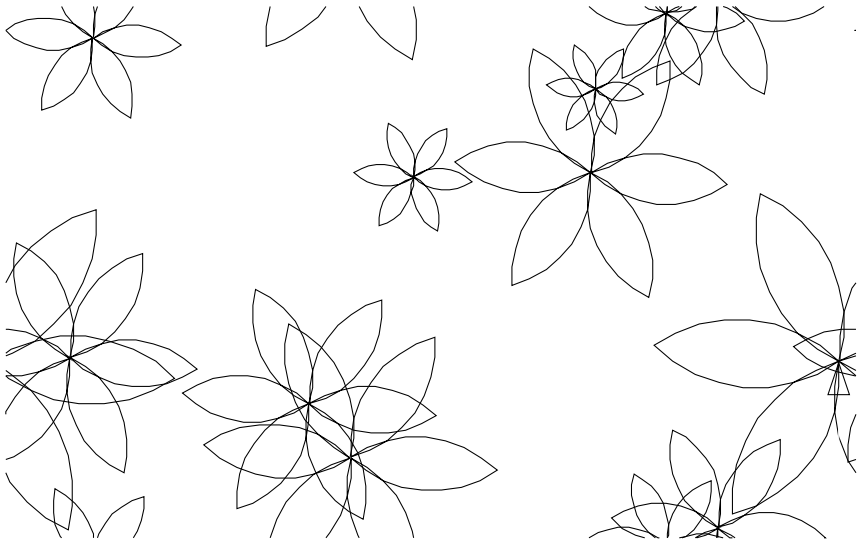
SATTUMA-sanalla voidaan myös vaihdella kukkien kokoa:

```
10 KERTAA 640 SATTUMA 400 SATTUMA PAIKKA 20 SATTUMA KUKKA VIELÄ?
```

Nyt koko koodin pitäisi näyttää tältä:

```
KONNA ASEMA NÄYTÄ PINO NÄYTÄ

LUO :KOKO KAARI
    8 KERTAA :KOKO ETEEN 10 OIKEA VIELÄ?
VALMIS

LUO :KOKO TERÄLEHTI
    2 KERTAA :KOKO KAARI 8 * 10 = VASEN 180 VASEN VIELÄ?
VALMIS

LUO :KOKO KUKKA
    6 KERTAA :KOKO TERÄLEHTI 60 OIKEA VIELÄ?
VALMIS

10 KERTAA 640 SATTUMA 400 SATTUMA PAIKKA 20 SATTUMA KUKKA VIELÄ?
```

Syntyvän kuvan pitäisi taas vastata tätä satunnaisten sijaintien ja kokojen puitteissa:



B.8 Lisätehtäviä

- Kykenetkö tekemään kukan, jolla on eri määrä terälehtiä? (Vinkki: `KUKKA`-sanan toistojen määrä ja sen sisällä käännyttävä kulma)
- Kykenetkö tekemään kukan, jonka terälehdet ovat kapeammat tai leveämmät? (Vinkki: `KAARI`-sanassa käännyttävä kulma, muista päivittää `TERÄLEHTI`-sana)
- Kykenetkö tekemään terälehtien määrästä ja pulleudesta satunnaisen kedon kukissa?