Beta

Ohjelmointia Scratchin kanssa

Matti Nelimarkka – Noora Vainio – Nyyti Kinnunen
Aiantasaistanut Ville Timonen

2015 (2011)

Kuulemme mielellämme mielipiteitäsi materiaalista. Mitä voisimme parantaa, miten asiat voisi ilmaista selkeämmin – mihin itse käytät tätä materiaalia.

Lähetä palautteesti sähköpostitse matti.nelimarkka@hiit.fi

Työn valmistumista ovat tukeneet:

NOKIA

KERHOKESKUS

HELSINKI INSTITUTE FOR INFORMATION TECHNOLOGY

Hahmot ja animointi

Toistorakenteet

Ehtorakenteet

Multimedia

Muuttujat ja loogiset operaatiot

Kynä

Tapahtumapohjainen ohjelmointi

Esipuhe

Perusopetuksen vuoden 2004 opetussuunnitelmassa yhtenä aihekokonaisuutena on *ihminen ja teknologia*. Tämän oppiainekokonaisuuden keskeisenä sisältönä on *tietotekniikan käyttäminen* ja *teknologian ymmärtäminen.* Lisäksi kannustetaan *kehittämään välineiden, laitteiden ja koneiden toimintaperiaatteiden ymmärtämistä.* Opetussuunitelmassa ei kuitenkaan painoteta tietokoneiden toimintaa. Tämä tarkoittaa sitä, etteivät oppilaat välttämättä osaa vastata keskeiseen kysymykseen *miten tietokoneet toimiva*t.

On luonnollista, että tietotekniikan käyttö on nostettu esiin myös opetussuunnitelmassa. Tietotekniikka on jo nyt merkittävä osa yhteiskuntaa, ja sen rooli tulee kasvamaan. Käytämme sitä nykyisin myös huomaamattamme - sinunkin taskustasi voi löytyä tietokone.

Oppilaiden tulisi mielestämme perinteisen tietotekniikan käytön lisäksi myös ymmärtää *jotain* sen toiminnasta. Mutta mitä tietotekniikka oikeastaan on? Peter J. Denningin (2003) mukaan se on esimerkiksi tietokantoja, tietorakenteita ja käyttöliittymien suunnittelua. Käytännössä näiden teemojen kanssa päädytään usein käytännön toteutuksen kysymyksiin, joissa algoritmien suunnittelu ja niiden toteutus eli ohjelmointi on keskeistä.

Tietokoneet ovat enemmän kuin valmiita välineitä, jollaisina ne tavallisesti esitetään. Pyrimme tämän materiaalin avulla tuomaan esille toisen näkökulman tietokoneiden käyttöön. Haluamme korostaa, että tietotekniikka voi olla myös **ongelmanratkaisun väline**. Toki myös valmiit ohjelmat, kuten taulukkolaskenta, tarjoavat välineitä tähän.

Kuitenkin, ohjelmoinnin avulla on mahdollista ratkaista erilaisia ongelmia sekä kehittää ongelmanratkaisun ajattelutaitoja systemaattisimmiksi.

Toisin kuin ihmiset, tietokoneet eivät osaa tehdä päätelmiä tai muuttaa ongelmanratkaisustrategiaansa kesken ohjelman suorituksen. Siksi ongelmanratkaisustrategia tulee esittää täsmällisemmin, käyttäen formaalia, etukäteen määriteltyä kieltä. Tätä hyödyntäen tietotekniikka voi auttaa ongelmanratkaisutaitojen kehityksessä.

Työvälineeksi tässä materiaalissa olemme valinneet Scratch-nimisen ohjelmointiympäristön. Scratch on MIT-yliopistolla kehitetty lapsille suunnattu ohjelma, jonka avulla voi tutustua algoritmiseen ajatteluun innostavalla tavalla. Innostusta auttaa esimerkiksi visuaaliset käyttöliittymäelementit, jotka rohkaisevat lapsia kokeilemaan ja lisäävät ohjelmoinnin mielekkyyttä. Lisäksi Scratchillä oppii perusteet nopeasti, minkä jälkeen on helpompaa siirtyä käyttämään kehittyneempiä ohjelmointikieliä.

Lopuksi on myös syytä muistaa, että ohjelmointi on väline myös luovuuteen, ei vain ongelmien ratkaisemiseen. Tämä tulee pitää mielessä myös kerhon toiminnassa, vaikka olemme materiaalissa korostaneet ongelmanratkaisun keskeistä roolia. Luovan toiminnan ja onnistumisen kokemuksien kautta syntyy oma halu oppia enemmän. Lisäksi arkipäiväistymisen myötä tietotekniikka on muuttunut vähemmän vakavaksi - se voi olla myös hauskaa.

Alkuun Scratchin kanssa

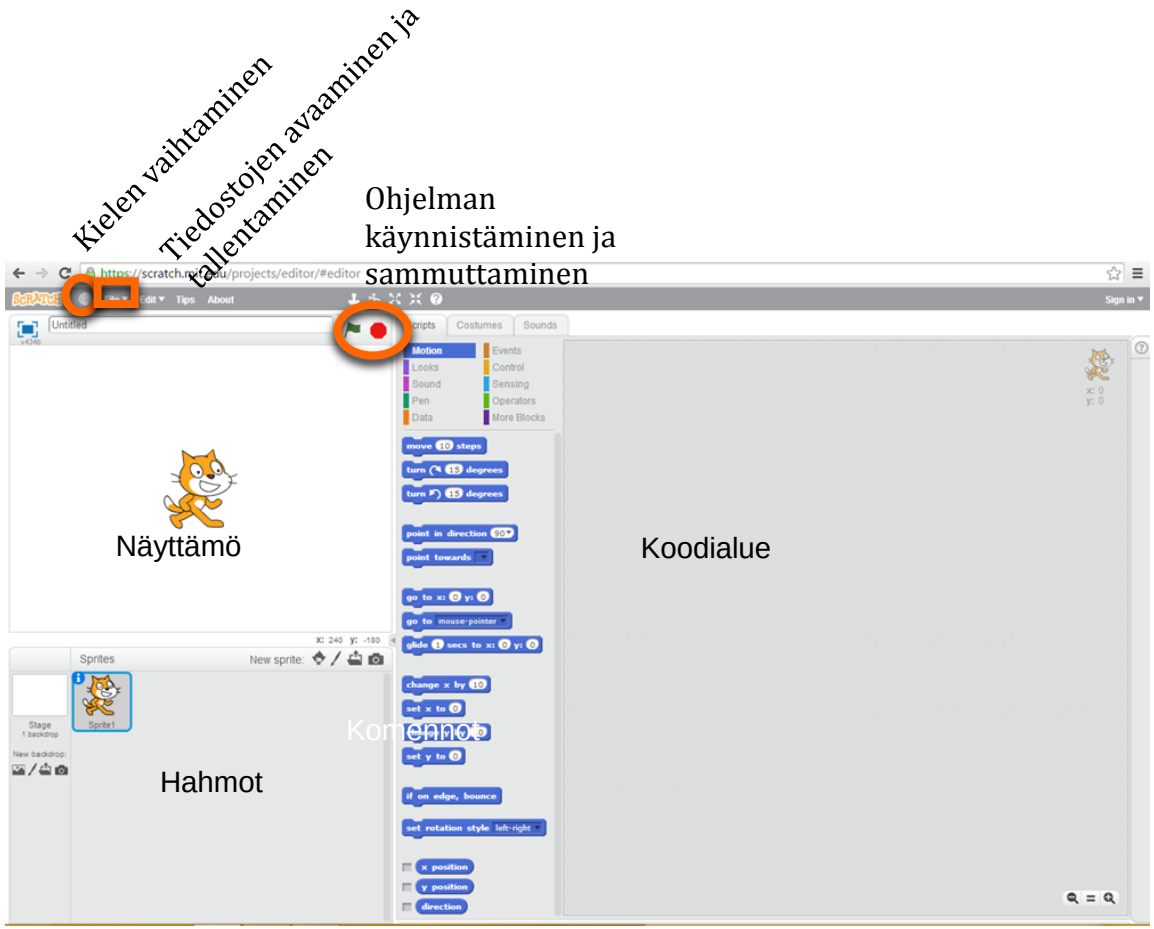
Voit käyttää Scratchia osoitteesta: http://scratch.mit.edu/. Tämä opas on suunniteltu englannin kielistä Scratchia käyttäville. Kielen voit vaihtaa ohjelman yläreunassa olevasta maapallon kuvasta. Mikään ei siis estä sinua opettamasta suomeksi.

Scratch-ikkuna muodostuu käytännössä neljästä eri alueesta. Keskimmäisin laatikko sisältää kaikki komennot. Oikealla on alue johon valitut komennot raahataan. Komennot raahataan hiiren osoittimella valikoista koodialueelle.

Jos klikkaat hiiren oikealla näppäimellä keskialuetta, saat valikon josta voit siistiä aluetta tai lisätä koodia selittävän kommentin. Kommenttien lisääminen on kannattavaa, jos ohjelma on monimutkainen.

Klikkaamalla hiiren oikealla näppäimellä yksittäistä koodilohkoa, saat valikon, josta saat ohjeita tätä lohkoa koskien, voit poistaa osan tai tehdä kopion osasta. Voit poistaa koodia myös siirtämällä sen pois keskialueelta. Koodi käynnistetään tuplaklikkaamalla sitä.

Vasemmalla yläreunassa on näyttämö eli stage, josta näet ohjelmasi suorituksen. Sen alapuolella on hahmo-valikko, josta näet luomasi hahmot ja voit luoda uusi hahmoja.



Tässä kappaleessa luodaan pohjaa tulevalle Scrach-ohjelmoinnille. Ensiksi käsitellään hahmoja muokkausta ja lisäämistä Scratchiin. Lisäksi esitetään yksinkertainen hahmojen liikutus. Tärkeää on ymmärtää, että yksittäiset Scracth-hahmot ovat toisistaan riippumattomia.

Scratchissa kaikki ohjelmat perustuvat hahmoihin ja niiden toimintaan. Yhdessä ohjelmassa voi olla useita hahmoja ja hahmot voivat myös kommunikoida keskenään.

Jokaisella hahmolla on omat ohjelmakoodista (scripts) , joiden mukaan ne käyttäytyvät toistaan välittämättä. Hahmot voivat myös kommunikoida keskenään ja niiden toimintaan voi vaikuttaa esimerkiksi toisten hahmojen sijainnit.

Hahmoja voidaan animoida luomalla niille ohjelmakoodeja, jotka liikkuttelevat niitä ja muuttavat niiden ulkonäköä. Hahmojen liikuttamiseen komentoja löytyy *motion*-valikosta. Hahmojen ulkonäköön liittyviä komentoja löytyy *looks*-valikosta.

Luodessasi uuden projektin, siihen lisätään automaattisesti yksi kissa-hahmo. Tämän hahmon voi kuitenkin poistaa painamalla hiiren oikealla näppäimellä näyttämön alapuolella olevaa hahmon kuvaketta ja valitsemalla *delete.*

Uuden hahmon voit luoda joko valitsemalla jonkin Scratchin valmiista hahmoista tai piirtämällä kokonaan uuden hahmon. Voit myös valita satunnaisen hahmon, joka lisätään projektiisi.

Myös valmiita hahmoja voi muokata. Valitsemalla hahmon valikosta *costumes*-osion voit muokata hahmon asua. Täällä voit myös lisätä hahmolle uusia asuja. Uusia asuja lisätään joko piirtämällä kokonaan uusi asu tai tuomalla valmiita asuja. Nämä asut eivät ole uusia hahmoja, vaan hahmosi voi muuttaa ulkonäköään *switch to costume*- tai *next costume*-komennolla.

Voit myös muokata taustaa ja sille voidaan luoda omia ohjelmakoodeja. Taustaa pääset muokkaamaan klikkaamalla pienoiskuvaa taustasta hahmo-valikossa.

Taustan ohjelmakoodeilla voit muokata taustaa, joko jatkuvasti tai siihen voivat vaikuttaa esimerkiksi hahmojen sijaintien perusteella tai näppäin- ja hiirikomentojen avulla.

Samoin kuin hahmoille, taustallekin voidaan luoda uusia asuja. Nämä asut toimivat samoin kuin hahmojen asut, eli niitä voidaan ohjelman ajon edetessä vaihtaa.

Scratch-ohjelmointi perustuu erilaisiin hahmoihin. Hahmoja voi luoda yhteen ohjelmaan useita ja ne toimivat oletuksena toisistaan riippumatta. Hahmot voivat myös kommunikoida keskenään tai muutoin vaikuttaa toisiinsa.

Hahmoja voi liikuttaa ja niiden ulkoasua voi muokata. Myös taustaa voi muokata ja sille voi luoda toiminnallisuuksia.

Toistorakenteet

Oppimistavoite tämän kappaleen osalta on osoittaa, kuinka ohjelman rakenteita voi toistaa muutenkin kuin kopioimalla. Tähän tarkoitukseen esitetään kaksi erilaista toistorakennetta, for- ja while-rakenteet.

Oppimistavoitteena on ymmärtää, milloin for-rakennetta tulee käyttää ja milloin taas while-rakenne on oikea. Tämän takia kannattaa valita tehtäviä siten, että ne pakottavat pohtimaan ja tekemään valintoja näiden kahden rakenteen välillä.

Joskus tarvitsemme usean kopion samasta paperista. Sen sijaan, että painamme kopiointinappulaa hyvin monta kertaa kerromme kopiokoneelle, että tarvitsemme näin monta kopiota. Ohjelmointikielissäkin on joskus tarpeen toistaa tiettyä rakennetta, ja tämän takia kielissä on yleisesti toistorakenteita.

Ensimmäinen toistorakenne, jota kutsutaan **for-rakenteeksi**, toimiii tilanteessa, jossa tiedät etukäteen kuinka monta kertaa toiston tulee tapahtua. Scratch-ympäristössä rakenne löytyy *Control*-valikon alta repeat-rakenteena. Rakenteelle kerrotaan, kuinka monta kertaa toiston tulisi tapahtua.

Sivussa on selvästi toistettu tiettyä rakennetta useamman kerran. Tämän takia se voidaan ilmaista yksinkertaisemmin käyttäen for-rakennetta.

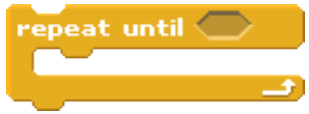
=

Toistorakenteen käyttö selkeyttää selvästi ohjelman koodia. Koodi on lyhyempi ja ilmaisee selkeämmin mitä se tekee verrattuna oikeanpuoleiseen esimerkkiin.

Tehtävä: Kissa syö kissanminttua, mutta voi pahoin tämän jälkeen. Pahoinvoivana kissa vaihtaa väriään kymmenen kertaa.

Koska tiedetään etukäteen varmasti, kuinka monta toistoa tulee tapahtumaan, for-rakenne on oikea väline tämän ongelman ratkaisemiseksi:

Jos et etukäteen tiedä, kuinka monta kertaa rakennetta tulisi toistaa, tarvitsemme erilaisen rakenteen. Tätä rakennetta kutsutaan **while-rakenteeksi**. Scratchissä while-rakenne löytyy *Control*-valikosta komentona repeat until.



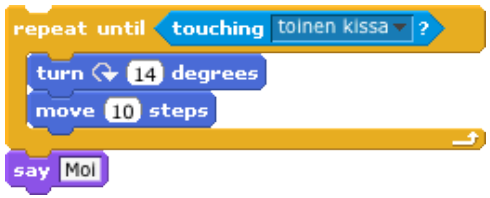
While-rakenteen sisälle asetetaan totuusarvon saava ehto. Totuusarvoista olennaista on, että se on kaikissa tilanteissa joko tosi tai epätosi. Jos se on tosi, while-rakenteen suoritusta jatketaan, jos se on epätosi jatketaan muun ohjelman suoritusta. Esimerkiksi *Sensing*-valikosta löytyy totuusarvoja palauttavia ehtoja. Esimerkiksi siellä on palikka, joka kertoo koskeeko hahmo reunaa vai ei. Sen arvo siis on tosi, kun hahmo koskettaa reunaa ja kun hahmo ei kosketa reunaa, sen arvo on epätosi.

Tehtävä: Tee ohjelma, jossa kissa liikkuu kunnes se koskee seinää.

Ohjelman suoritusta jatketaan kunnes kissa koskee seinään. Etukäteen ei tiedetä, kuinka monta liikuttamista tähän vaaditaan, jonka takia tarvitaan while-rakennetta.

Tehtävä: Kaksi kissaa rakastavat toisiaan ja haluavat olla toistensa kanssa – valitettavasti niiden suunnistustaidot ovat heikot, joten he vain kääntyvät ympäriinsä ja menevät eteenpäin. Jos he pääsevät toistensa lähellä, niin he pysähtyvät.

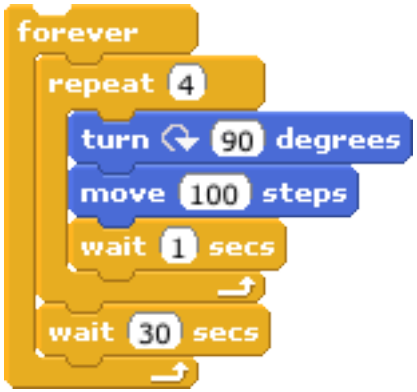
Nyt tehtävässä tarvitaan kaksi eri hahmoa, mutta molemmille tulee samanlainen koodi. Etukäteen ei tiedetä kuinka kaukana kissat ovat, joten toistojen määrää ei tunneta. Siksi on syytä käyttää while-rakennetta. While-rakenteen lopetusehtona on se, että toista kissaa kosketetaan.



Kolmas erikoistyyppi toistorakenteesta on loputon toisto. Käytännössä tämä on while-rakenne sellaisessa tapauksessa, jossa ehto on aina tosi – tällöin toistettavaa rakennetta toistetaan kunnes ohjelman suoritus keskeytetään. Scrath-ympäristössä tätä kutsutaan forever-rakenteksi . Huomaa, että voit keskeyttää ohjelman suorittamisen oikean yläkulman punaisella kahdeksankulmiolla.

Tehtävä: Kissa vartioi linnaa ja kiertää sen ympäri puolen minuutin välein.

Tässä tehtävässä yhdistetään kaksi erilaista toistorakennetta. Sisempi for-rakenne siirtää kissaa linnan ympäri kun taas ulompi toistaa tätä kiertoliikettä puolen minuutin välein. Tässä tulee olla erityisen tarkkaavainen siinä, mitkä elementit kuuluvat mihinkin toistorakenteeseen. Ensiksi kannattaa ratkaista osa-ongelmana 'kierretään linnan ympäri' ja tämän jälkeen ratkaista while-rakennetta sen ympärille.



Ohjelmoinnissa on usein tarpeen toistaa samaa rakennetta useamman kerran. Leikkaa-liimaa-kopio on toki tehokas keino tähän, mutta johtaa lopulta vaikeasti ylläpidettäviin ohjelmiin ja turhaan toistoon. Tämän välttämiseksi on olemassa kaksi eri rakennetta:

For-rakenne on tarkoitettu käytettäväksi silloin kun tiedetään täsmälleen, kuinka monta kertaa asia halutaan toistaa.

While-rakenne on tarkoitettu tilanteisiin, jossa ei etukäteen tiedetä toistokertojen määrää.

Toisin kuin for-rakenne, while-rakenne tarvitsee aina ehdon kuinka kauan rakennetta toistetaan. Tämä ehto on totuusarvo, eli aina joko tosi tai epätosi. Jos se on tosi, suoritusta jatketaan ja jos se on epätosi, suoritus keskeytetään.

Ehtorakenteet

Kappaleen oppimistavoitteena on ymmärtää ehtolauseiden merkitys ohjelmoinnissa. Oppilaan on havaittava, että tietyt suoritukset ovat vaihtoehtoisia tai ne tehdään vain tietyn ehdon täyttyessä. Tulee siis ymmärtää totuuslauseiden perusteet ja vaihtoehtoiset suoritustavat.

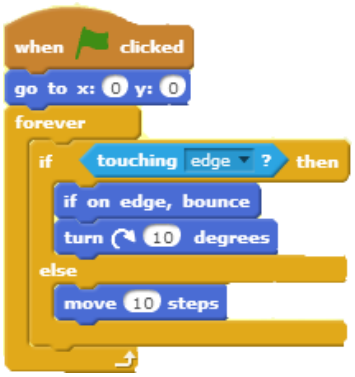
Arkielämässämme usein teemme päätöksien ehtojen avulla. Menemme tien yli jos autoja ei tule, muutoin odotamme, että autot ohittavat tien. Tai, menemme ulos jos ei sada, muutoin jäämme sisälle.

Ohjelmoinnissa on usein tarpeen asettaa ohjelmalle ehtoja joiden mukaan se toimii. Esimerkiksi hahmon koskettaessa reunaa se saatetaan haluta palauttaa takaisin keskelle tai vaihtoehtoisesti pysäyttää hahmon liike. Tämä toteutetaan ehtolauseilla.

Ehtorakenteet ovat tärkeä osa ohjelmointia. Niiden avulla voidaan jaksottaa ohjelman kulkua ja jakaa ohjelman kulku eri osiin tilanteesta riippuen. Ehtorakenteet ovat if-lause ja else-lause. Scratchissa ehtorakenteet löytyvät *Control*-valikon alta.

Tehtävä: Kissa haluaa valvoa näyttämöä. Tee ohjelma, jolla kissa käy mahdollisimman suuren alueen kentästä läpi.

Toistetaan loputtomasti seuraavaa ohjetta: jos kosketetaan seinään, kimpoa ja käänny kymmenen astetta, muutoin liiku kymmenen askelta suuntaan, jonne osoitan. Näin kissa käy hitaasti, mutta varmasti koko näyttämön läpi. Aina seinän tullessa vastaan se kääntyy ja muutoin se liikkuu eteenpäin.



Ehtolauseiden avulla ohjelman suoritusta voidaan jakaa osiin. Osat suoritetaan ehtojen toteutuessa.

If-lauseen sisällä oleva osa suoritetaan if-lauseen ehdon toteutuessa, **else-lauseen** sisällä oleva osa suoritetaan jos ja vain jos if-lauseen ehto ei toteudu. If-lause voi esiintyä joko else-lauseen kanssa tai ilman sitä.

Tämä kappale esittää perusteet äänen liittämisestä Scratch-ohjelmaan. Tavoite on, että tämän kappaleen myötä oppilas osaisi liittää yksinkertaisia ääniä osaksi ohjelmaansa. Äänen liittämisen kautta on myös mahdollista saavuttaa uusia ilmaisun muotoja. Esimerkiksi on mahdollista äänittää rytmejä tai tunnettuja lauluja ja ohjelmoida niihin sopiva animaatio.

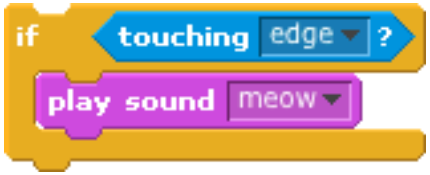
Maailmamme havainnot perustuvat useisiin erilaisiin aisteihin. Esimerkiksi televisio-ohjelmissa ääni on vähintään yhtä tärkeä osa kokemusta kuin näköaistiin perustuva kokemus. Myös tietokonepeleissä sekä yhä enemmän muissakin ohjelmissa musiikki ja äänet ovat keskeinen osa käyttökokemusta.

Myös Scratchissä on mahdollista liittää ääniä tapahtumiin. Tämä tapahtuu *Sound*-valikon kautta. Yksinkertaisimmillaan äänen toistaminen on valmiiksi annetun äänitiedoston toistamista. Oppilas voi myös nauhoittaa oman äänensä tähän valitsemalla *Record* soitettavien äänien valikoimasta.

On myös mahdollista tehdä yksinkertaisempia yksittäisistä äänistä koostuvia soitoksia. Tähän auttavat yksittäisten tahtien soittamiseen tarkoitetut komennot

Tehtävä: Kun kissa törmää seinään, niin sitä sattuu ja se sanoo Mau.

Ehtolauseen sisälle laitetaan nyt komento, joka toistaa ääntä.



Matalamman tason äänen käsittelystä esimerkkinä toimii seuraava musiikkiesitys. Kun vihreää lippua painetaan, niin toistuu yksinkertainen rytmi.

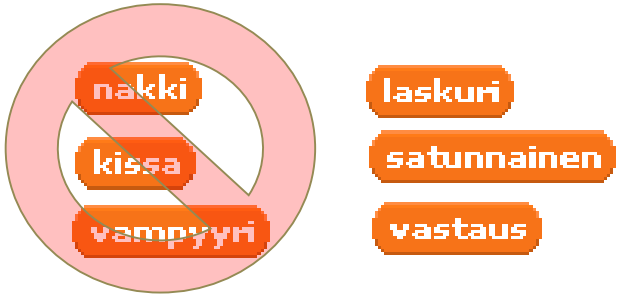
Audiovisuaalinen kokemus voi olla mielekäs ja parantaa ohjelman käytettävyyttä ja käyttökokemusta. Scratchissä voi soittaa ääntä kahdella eri tavalla: korkealla tasolla suoraan äänitiedostojen kautta tai matalammalla tasolla yksittäisiä nuotteja käyttäen.

Muuttujat

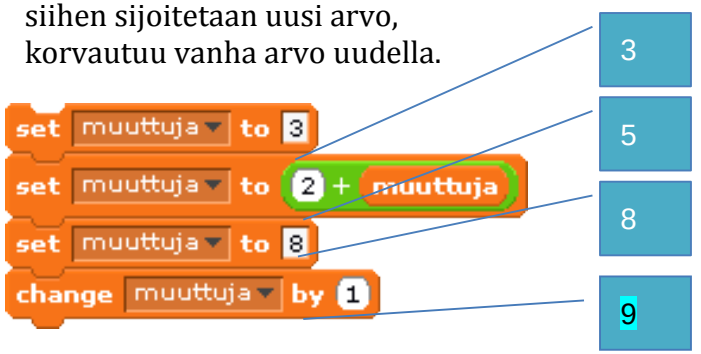
Kappaleen kannalta tärkeintä on, että oppilas ymmärtää muuttujien käsitteen. Lisäksi tavoitteena on oppia käyttämään muuttujia ja soveltaa niitä erilaisissa tilanteissa. Käyttö sisältää myös operaatiot, joita muuttujille voidaan tehdä. Lisäksi harjoitellaan listojen käyttöä

Usein ohjelmoinnissa tulee tarve tallentaa tietoa ohjelman käyttöön, esimerkiksi pisteenlaskua varten. Ohjelmoinnissa tämä on toteutettu muuttujien avulla. Muuttujat ovat tietokoneen muistista varattuja muistipaikkoja, joihin päästään tämän nimen kautta. Niihin voidaan tallentaa niin sanoja kuin lukujakin.

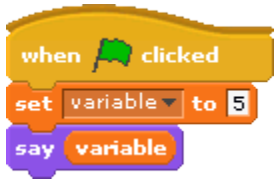
Koska muuttujat ovat tiettyjä muistipaikkoja tietokoneen muistissa, on jokainen muuttuja nimettävä. Näin tietokone tietää, minkä muuttujan arvoa halutaan käyttää. Tästä syystä kahdella muuttujalla ei voi olla samaa nimeä, sillä muuten tietokone ei tietäisi, kumpaa muuttujaa on käytettävä. Muuttujat kannattaa nimetä muuttujien käyttötarkoituksia kuvaaviksi. Esimerkiksi jos muuttuja kuvaa yhteenlaskun vastausta, on se järkevää nimetä summaksi eikä esimerkiksi maksalaatikoksi. Kun muuttujien määrä kasvaa, on järjettömästi nimettyjen muuttujien hallitseminen hyvin vaikeaa ja virhealtista.



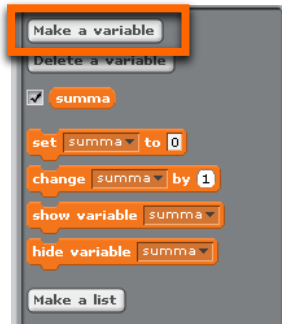
Koska muuttujat ovat varattuja muistipaikkoja, ei niille kerran annettu arvo ole lopullinen vaan muistipaikassa oleva arvo voidaan korvata uudella. Näin voidaan luoda esimerkiksi laskureita. Muuttujalla voi olla vain yksi arvo kerrallaan. Mikäli vanhaa arvoa muutetaan tai siihen sijoitetaan uusi arvo, korvautuu vanha arvo uudella.

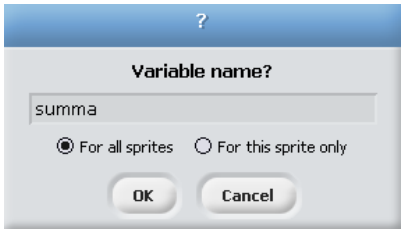


Muuttujan nimi ja *muuttujan arvo ovat kaksi eri asiaa*. Muuttujan nimeä ei voi muuttaa sen luomisen jälkeen. Muuttujan arvo sen sijaan tarkoittaa lukua tai sanaa, jonka muuttuja sisältää. Viereisessä koodissa muuttujan “variable” arvoksi asetetaan 5. Tämän jälkeen ohjelmaa käsketään sanoa ”variable”. Kun ohjelma ajetaan, kissa sanoo 5 eikä variable, sillä muuttujan arvo on 5.



Scratchissa muuttuja luodaan *Data*-valikosta löytyvästä *Make a variable* -painikkeesta. Muuttujalle on annettava nimi, minkä jälkeen se ilmestyy valikkoon. Muuttujaa nimetessä voit valita voivatko muuttujaa käyttää kaikki hahmot vai vain sillä hetkellä valittu hahmo. Muuttujan luonnin jälkeen se näkyy myös näyttämön ylälaidassa.

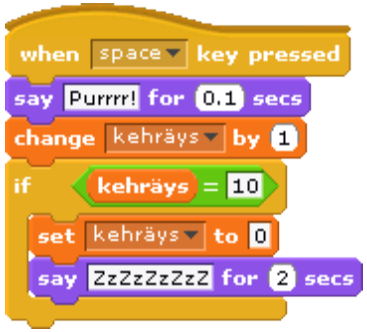




Muuttujalle asetetaan arvo lisäämällä ohjelmaan *set to*-palikka. Keskellä olevasta valikosta voit valita muuttujan, jonka arvoa muutetaan. Muuttujan arvoa voidaan muuttaa monta kertaa, jolloin viimeisin muutos jää voimaan.

Muuttujan arvoa voidaan muuttaa myös niin, että sen arvoa kasvatetaan tai pienennetään tietyn verran. Tähän käytetään *change by*-palikkaa. Tämä on kätevää silloin, kun muuttujan arvoa muutetaan usein, eikä muuttujan arvoa tiedetä tai sitä haluta asettaa tiettyyn arvoon – esimerkiksi laskureissa.

Tehtävä: Kissa kehrää kun sitä silittää (välilyöntiä painamalla). Kun kissaa on silittänyt kymmenen kertaa, se menee nukkumaan. Nukuttuaan kissa on taas valmis kehräämään.

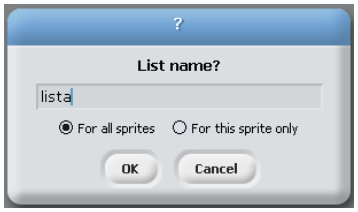


Tehtävässä pitää laskea, montako kertaa välilyöntiä on painettu. Tätä varten luodaan muuttuja kehräys, joka kasvaa yhdellä aina, kun välilyöntiä painetaan. Kun välilyöntiä on painettu 10 kertaa – eli kun muuttujan kehräys arvo on kymmenen - kissa menee nukkumaan.

Joskus ohjelmoinnissa tulee tarve tallentaa useita arvoja, jotka liittyvät samaan asiaan. Tällöin ei ole järkevää tehdä useita muuttujia, sillä niitä voitaisiin joutua luomaan useita kymmeniä. Tämä on ratkaistu ohjelmoinnissa listojen avulla. Esimerkiksi voi olla samaan asiaan liittyviä käyttäjän vastauksia. Tällöin sen sijaan, että niistä tehtäisiin jokaisesta oma muuttujansa, on mielekkäämpää, että ne ovat yhdessä listassa.

Listoissa tietoa laitetaan peräkkäin. Niitä voi ajatella jonoina, joissa jokaisella alkiolla on järjestysnumero. Numerointi alkaa ykkösestä ja jatkuu alkioiden määrän verran. Mikäli lista on viiden alkion pituinen, ei listan kohtaan 15 voi lisätä mitään, vaan se aiheuttaa virheen, joka ilmenee skriptin punaisena ympäröintinä.

Lista luodaan *Data*-valikosta löytyvästä *Make a list* -painikkeesta. Listalle on annettava nimi, minkä jälkeen se ilmestyy valikkoon. Listaa nimetessä voi valita voivatko listaa käyttää kaikki spritet vai vain sillä hetkellä valittu. Listan luonnin jälkeen se näkyy stagen ylälaidassa.



Listaan voidaan lisätä alkio *add* –palikalla. Tällöin lisätty asia menee listan perälle ja listan koko kasvaa yhdellä. Listaan voi lisätä numeroita, merkkijonoja ja muuttujia.

Kuten muuttujia, myös listan alkioiden arvoja voi muuttaa *Replace item*-palikan avulla. Tällöin alkion vanha arvo korvataan uudella.

Alkiota voidaan myös poistaa listasta. Tällöin valittu alkio poistetaan ja kaikkien sen alla olevien alkioiden järjestysluku pienenee yhdellä, jolloin myös listan pituus pienenee yhdellä.

Myös listan keskelle voi lisätä uuden alkion. Tällöin valittuun kohtaan sijoitetaan uusi haluttu arvo ja kaikkien sen alapuolella olevien alkioiden järjestysluku kasvaa yhdellä, jolloin myös listan pituus kasvaa yhdellä.

Sellaisinaan muuttujat toimivat lähinnä tiedon säilyttämisessä. Tilanne on samanlainen kuin kirja, jonka voi lukea, mutta saatua tietoa ei voi soveltaa mihinkään. Mikäli muuttujia voitaisiin käyttää vain tiedon tallentamiseen, ei laskutoimitusten suorittaminen olisi mahdollista.

Varsin useat pelit vaativat esimerkiksi yhteenlaskua pisteiden laskemiseen. Tämän takia ohjelmoinnissa on käytössä kaikki perusmatemaattiset operaatiot: yhteen-, vähennys, kerto- ja jakolasku sekä modulo eli jakojäännös.

Lisäksi Scratchissa on palikka, joka sisältää monimutkaisempia matemaattisia operaatioita, kuten sini, tangentti, neliöjuuri ja logaritmi. Round pyöristää sille annetun luvun normaalien pyöristyssääntöjen mukaan. Ne löytyvät *operators*-valikon alta.

Voit operaattorien avulla laskea muuttujien arvoilla peruslaskuja niin että toinen, molemmat tai ei kumpikaan luvuista on muuttuja. Operaattoreja voi myös yhdistää keskenään.

Tämän lisäksi muuttujilla toimivat pienempää kuin, suurempaa kuin ja yhtä suurta kuin operaatio, joihin tutustutaan seuraavassa luvussa.

Operators-valikosta löytyy myös toimintoja merkkijonojen yhdistämiseen, läpikäymiseen sekä pituuden selvittämiseen.

Join yhdistää sille annetut merkkijonot yhdeksi merkkijonoksi. Yhdistetty merkkijono ei tallennu mihinkään automaattisesti, vaan se pitää muistaa tallentaa muuttujaan.

Letter n of palauttaa sille annetun sanan n:nen kirjaimen ja length of palauttaa sille annetun sanan pituuden.

Tehtävä: Kissa on kerännyt kaupasta ostettavat asiat listaan. Tee ohjelma, jolla kissa kertaa, mitä hän aikoo ostaa kaupasta.

Koska listassa on useita alkioita, joille kaikille halutaan tehdä sama toiminto, kannattaa käyttää toistorakennetta.

Koska listan pituus ei ole vakio käytetään toistojen määränä *length of*- palikkaa, joka palauttaa listan pituuden. Näin jokainen listan alkio käydään kerran läpi. *Item of* –palikka palauttaa alkion arvon, tässä tapauksessa sanotaan aina alkio kohdassa i, joka kasvaa jokaisella toistokerralla yhdellä.

Muuttujat mahdollistavat monimutkaiste ohjelmien tekemisen. Muuttujin pystytään tallentamaan ohjelma tilaa ja niitä pystytään muuttamaan ohjelman suorituksen aikana.

Muuttujia on kahdenlaisia: yksittäisiä muuttujia ja muuttujien kokonaisuuksia, listoja.

Muuttujille voidaan suorittaa perusoperaatioita, kuten yhteen-- ja vähennyslaskua tai sanasta tutkia, mikä on sen n:nnes kirjain.

Loogiset operaatiot

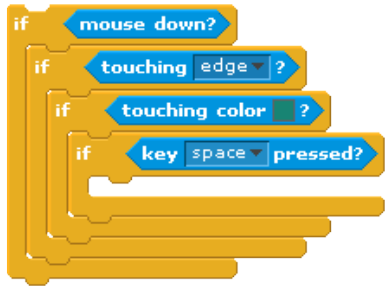
Kappaleessa käsitellään loogisia ehtolausekkeita, kuten pienempää kuin ja yhtä suuri. Lisäksi käsitellään loogisten ehtolausekkeiden yhdistämistä operaattoreilla ja lauseiden käyttöä ohjelmoinnissa.

Loogiset operaattorit saattavat kuulostaa vierailta, vaikka todellisuudessa niitä käytetään joka päivä. Tekemiselle saatetaan asettaa useita toisistaan riippumattomia ehtoja käyttäen sanoja ja, tai ja ei. Esimerkiksi ”Lähden illalla elokuviin jos saan siivottua ja tehtyä läksyt”. Joku voisi kuitenkin lähteä elokuviin jo kun on joko siivottu tai tehty läksyt. Myös ohjelmoinnissa on tarpeellista asettaa joidenkin koodinpätkien suorittamiselle ehtoja, joihin liittyy useampia muuttujia.

Ohjelmoinnissa on usein tarpeellista vertailla asioita ja tehdä asioita vain, jos tietty ehto täyttyy. Usein yksi ehto ei kuitenkaan riitä halutun lopputuloksen saamiseksi tai riittää että toinen ehdoista täyttyy.

Loogisten operaattorien avulla voidaan muodostaa kokonaisuuksia, jotka koostuvat pienemmistä osista, *lausekkeista*. Lausekkeet voivat olla muuttujia, niiden arvon tutkimista, hahmoihin liittyviä totuuksia tai muita loogisten operaattoreiden avulla muodostettuja lauseita. Muodostuneet kokonaisuudet voivat siis toimia myös lausekkeina.

Esimerkiksi halutaan, että hahmo kääntyy, mikäli se koskee vaaleanpunaista reunaa. Liikkumisen suorittamiseksi tarvitaan siis kaksi ehtoa. Ohjelman voisi toteuttaa laittamalla kaksi if-lausetta sisäkkäin, mutta kun ehtojen määrä kasvaa tulisi koodista vaikealukuista ja koodi kasvaa huomattavasti.



Tämä ongelma voidaan ratkaista loogisten operaattorien avulla. Niillä voidaan yhdistää ja vertailla erillisiä ehtoja ja muodostaa niistä kokonaisuus, jolla on vain yksi totuusarvo. Kun ehdoista muodostuu yksi totuusarvo, voidaan se laittaa sellaisenaan esimerkiksi if-lauseen ehdoksi. Alla and-operaattoria käyttävä toteutus yllä olevasta if-lause-hirviöstä.

Scratchista löytyy kuusi erilaista loogista operaattoria:

Yhtä suuri kuin	=	not	Ei
Pienempää kuin	<	or	Tai
Suurempaa kuin	>	and	Ja

Pienempää kuin, suurempaa kuin ja yhtä suurta kuin ovat monelle varmasti ennestään tuttuja ala-asteen matematiikasta. Ohjelmoinnissa vertailu operaattorit toimivat samalla tavalla.

Alla olevassa taulukossa on esimerkki vertailuja ja niiden palauttamat totuusarvot. Yleensä vertailuoperaattoreja käytettäessä ainakin toinen vertailtavista on muuttuja, jonka arvo muuttuu ohjelman suorituksen aikana.

2 < 5	Tosi
5 > 9	Epätosi
4 = 2	Epätosi
3 = 3	Tosi
5 < muuttuja	Riippuu muuttujan arvosta

And-operaattori ja or-operaattori eivät todennäköisesti ole ennestään tuttuja, mutta niiden toiminta on helppo ymmärtää.

And-operaattori on ‘tosi’, mikäli molemmat vertailtavat lauseet ovat tosia.

Or-operaattori on ‘tosi’, mikäli *vähintään* toinen vertailtavista lauseista on tosi. Tosi palautetaan siis myös, kun molemmat lausekkeet ovat tosia.

A	B	A and B	A or B
Tosi	Tosi	Tosi	Tosi
Tosi	Epätosi	Epätosi	Tosi
Epätosi	Tosi	Epätosi	Tosi
Epätosi	Epätosi	Epätosi	Epätosi

Ei-operaattorin arvo on sen sisälle laitetun lausekkeen totuusarvon vasta-arvon. Eli mikäli sisälle laitettu lauseke on tosi, on lauseen totuusarvo epätosi ja päinvastoin.

A	not A
Tosi	Epätosi
Epätosi	Tosi

Kuten alussa mainittiin, loogisia operaattoreja käytetään usein ehtolauseiden tai toistorakenteiden ehtoina.

If-lauseiden toistamisen sijasta käytetään and- or sekä not, mikäli ehtoja on useampia. Näin vältetään turhaa toistoa ja selvennetään ohjelman toimintaa.

Tehtävä
Kissa haluaisi pelata lautapeliä. Hän ei voi kuitenkaan pelata sitä yksin, joten hän tarvitsee molemmat lähellä asuvat ystävänsä Miisun ja Monnin. Kissan ystävät ovat arvaamattomalla tuulella ja heidän vastauksensa vaihtelevat. Kissan onkin suostuteltava molemmat kaverinsa pelaamaan kanssaan.

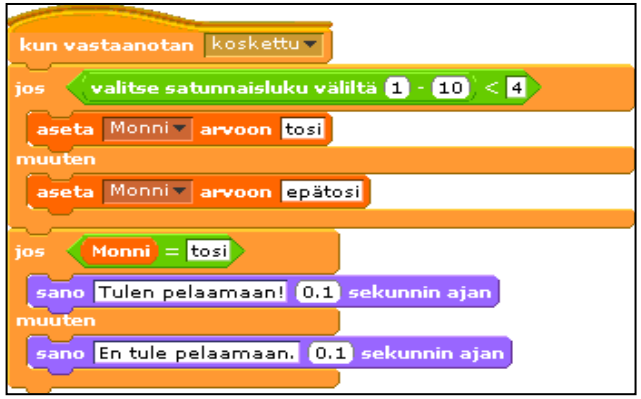
Tehtävässä on kaksi muuttujaa, Miisun suostuminen ja Monnin suostuminen, kun molemmat muuttujat ovat tosia, voidaan lautapeliä alkaa pelata. Miisun ja Monnin muuttuvat mielipiteet saadaan aikaan random-luvun avulla.

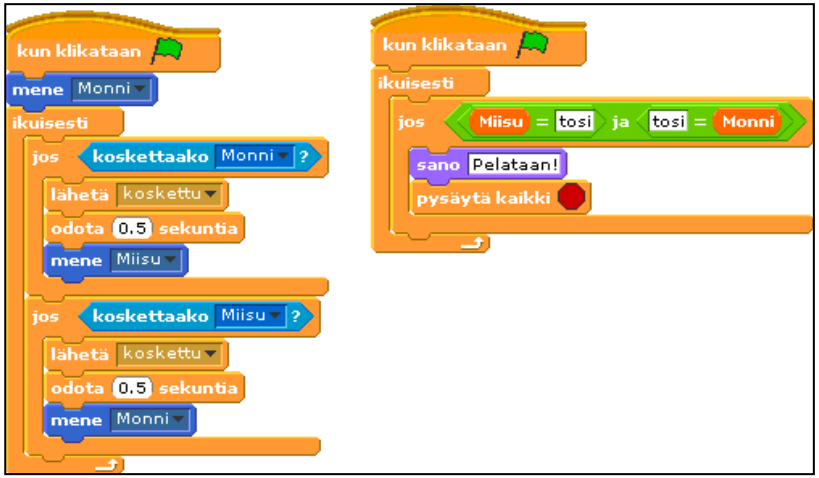
Mikäli random on väliltä1-3 vastaus on myönteinen ja väliltä 4-10 kielteinen. Kissan ystävät asuvat vastakkaisilla puolilla, joten kysyäkseen Kissan on juostava näyttämöä edestakaisin.

Monnilla ja Miisulla on molemmilla omat skriptinsä, jonka avulla ne vastaavat kissalle, ne ovat muuttujan nimiä lukuunottamatta samanlaisia. Kissalla pitää olla skripti liikkumiseen ja Miisun ja Monnin mielipiteen seuraamiseen.

Loogiset operaattorit tarjoavat mahdollisuuden vertailla ja muodostaa ehtolauseita muuttujien arvoihin liittyen. Loogisia operaattoreita ovat pienempää kuin, suurempaa kuin ja yhtä suuri kuin, jotka mahdollistavat arvojen vertailun.

Lisäksi loogiset operaatiot ja, tai sekä ei mahdollistavat täsmennetyn toiminnan loogisien lauseiden osalta. Esimerkiksi pienempää tai yhtäsuurta kuin kolme muodostuu kahdesta eri lauseesta: pienempää kuin sekä yhtäsuurta kuin.



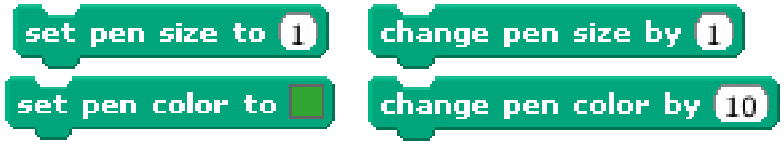


Kynä

Luvun oppimistavoite on oppia käyttämään suorempaa piirtotoiminnallisuutta. Piirtäminen tapahtuu hahmon liikutuksen avulla, määrittelemällä erikseen ollaanko nyt piirtotilassa vai ei.

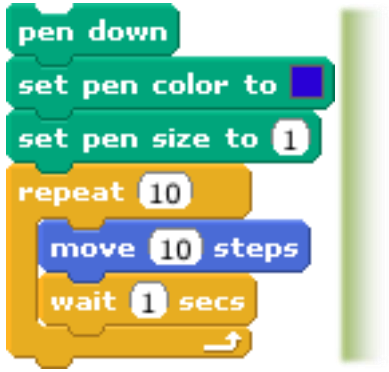
Sellaisenaan piirto-toiminnallisuuden sovellukset ovat varsin vähäisiä.. Se kuitenkin mahdollistaa aiemmin opittujen toisto- ja ehtorakenteiden käytön uudella tavalla, minkä takia tätä osuutta voikin käyttää hyvin aiemmin opittujen käsitteden kertaamiseen.

Tähän mennessä olemme käyttäneet leimasinta, valmiita kuvioita joita liikkuvat. Kuitenkin Scratch tarjoaa mahdollisuuden yksityiskohtaisempaan ja tarkempaan piirtämiseen *Pen*-valikon kautta. Kynän kokoa ja väriä voi muuttaa:

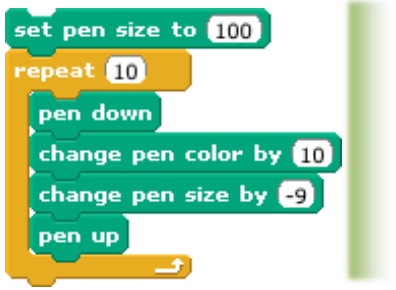


Kynä seuraa aina hahmoa. Kynä voi olla myös irti piirtopinnasta tai kiinni siinä. Piirtopinnassa kiinni ollessaan se piirtää, irti ollessaan ei piirrä.

Piirretään suora viiva käyttäen Scractchiä. Ensiksi asetetaan viivalle väri ja koko, tämän jälkeen liikkuessaan hahmon perässä on sininen viiva.



Nyt piirretään sisäkkäisiä isoja ympyröitä, joiden väri vaihtuu. Huomaa, että koon vaihtamiseen arvo voi olla myös negatiivinen.



Piirretään katkoviiva. Nyt ajatus on, että joka toisella askeleella kynä otetaan irti pinnasta ja askel otetaan jonka jälkeen kynä laitetaan takaisin pinnalle. Huomaa kuinka muuttujaa on käytetty laskemaan askelmäärää.

Scratchissä on mahdollista piirtää pinnalle suoraan käyttäen kynä-toiminnallisuutta. Kynälle voi määrittää sen koon ja värin.

Kynä piirtää silloin kun se on kiinni pinnassa ja ei piirrä kun se ei ole kiinni pinnassa.

Tapahtumapohjainen ohjelmointi

Tässä kappaleessa tarkoitus on oppia muodostamaan riippuvuussuhteita ohjelmien välillä. Riippuvuussuhteiden muodostaminen helpottaa ohjelmien luomista ja ylläpitämistä. Lisäksi se mahdollistaa uudet interaktiiviset muodot.

Keskeinen opetettava ajattelutapa on signaalin ja slotin käyttö tapahtumien abstraktoimisessa. Tavoite on, että oppilas osaa kappaleen lopussa aiheuttaa tapahtuman signaalilla ja ottaa tapahtuman vastaan jonkun toisen hahmot slot-rakenteella.

Usein arkielämässä ensimmäinen tapahtuma aiheuttaa toisen tapahtuman. Esimerkiksi ihminen ylittää autotien kun hän huomaa valon olevan vihreä. Tarkemmin siis, vihreä valo aiheuttaa sen, että ihminen ylittää tien, muutoin hän olisi vain pysähtyneenä.

Monimutkaisten ohjelmien tapauksessa saatetaan päätyä samanlaiseen tilanteeseen: joku toiminto aiheuttaa jotain muuta tapahtuvaksi ohjelmassa. Esimerkiksi Tallenna-napin painaminen aiheuttaa ohjelman tallentamisen. Kutsumme tätä tapahtumapohjaiseksi ohjelmoinniksi.

Yksinkertaisimmat tapaukset tapahtumapohjaisesta ohjelmoinnista on jo esitetty aiemmin. Esimerkiksi lipun painaminen ja välilyönti ovat tapahtumaan perustuvia toimintoja. Kun lippua painetaan, niin suoritetaan siihen liittyvä ohjelma. Kun välilyöntiä painetaan, niin suoritetaan siihen liittyvä ohjelma.

Kutsumme aiheuttavaa tapahtumaa **signaali**-termillä ja tapatumasta aiheutuvaa rakennetta **slot**-termillä. Esimerkissä signaaleja olisivat lipun tai välilyönnin painaminen ja slot olisi Scratchissä tähän signaaliin kytketty koodi.

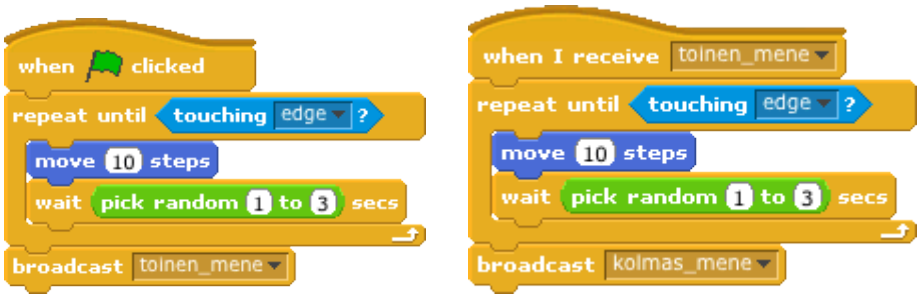
Nämä toki olivat yksinkertaisia esimerkkejä, mielenkiintoisempaa on kun hahmot voivat viestiä keskenään. Tämä tapahtuu lähettämällä signaali ja kirjottamalla sen vastaanottava slot. Nämä löytyvät *Control*-valikosta. 'broadcast' komento lähettää signaalin. Signaalille tulee valita nimi, jolla se tunnistetaan. Vastaavasti on 'when I receive' on slot-komento, johon liitetään slottiin kuuluva toiminto.

Tehtävä: Kaksi kissaa pelaavat hippaa. Aina kun kissa saa toisen kissan kiinni, niin kiinni saatu kissa sanoo "Hippa."

Esimerkissä toinen kissa etsii kissaa ja kun se saa kissan kiinni, niin lähettää viestin 'hippa'. Toinen kissa ottaa tämän viestin vastaan ja sanoo 'Olen hippa'.

Tehtävä: Kissat pelaavat viestiä, jossa seuraava kissa saa juosta vasta kun edellinen kissa on päässyt perille.

Koska emme etukäteen tiedä, kauanko kissalla menee juoksemiseen, niin joudutaan käyttämään tapahtumapohjaista ohjelmointia.



On oikeastaan olemassa kahdenlaisia signaaleja, synkroonisia ja asynkroonisia. **Asynkrooniset signaalit** eivät keskeytä ohjelman suoritusta slotin käsittelyn ajaksi, kun taas **synkrooniset singaalit** odottavat että slot saadaan suoritettua ja jatkavat sitten suoritusta.

Vasemman puoleinen koodi on asynkrooninen suoritus, eli "Hmmm" tulostuu näytölle heti kun taas oikean puoleisessa koodissa tulostus odottaa, että jotain tapahtuu-slot on saatu käsiteltyä.

Ohjelmoinnissa on tarpeen joskus esittää seuraussuhteita hahmojen välillä. Tätä mekaniikkaa kutsutaan signaaliksi ja slotiksi.

Signaali lähettää viestin. Viesti on Scratch-kielessä siis vain joku ilmiötä kuvaava sana tai lause.

Slot ottaa viestin vastaan. Slot-osuuteen kytketän, mitä hahmossa tulisi tapahtua signaalin vastaanottamisen jälkeen.

Tämä materiaali on ollut pintapuoleinen johdanto Scratch-ympäristöön ja ohjelmoinnin perusteisiin. MIT-yliopiston kehittämä Scratch-ympäristö on ollut laajemmin käytössä useissa kouluissa, ja sille on eritoten englanniksi saatavilla valmiita materiaaleja. Suosittelemme tutustumaan myös näihin, huomattavasti kattavempiin, materiaaleihin sekä Scratch-verkkosivuilla esiteltyihin projekteihin. Myös Scratchin ympärillä tapahtuu jatkuvasti tutkimustyötä, joka voi olla mielekästä seurata.

Kuitenkin sellaisenaan Scratch sopii innostuksen herättämiseen ja perusteiden opiskeluun. Kuitenkaan sillä ei voi tehdä varsinaisia tietokoneohjelmia. Seuraavana askeleena ohjelmoinnin saralla kannattaa siirtyä yleisesti käytettihin ohjelmointikieliin, vaikka niissä Scratchin visuaalinen ilmaisu puuttuukin.

Hyvä aloittelijakieli on esimerkiksi Python. Python on yksinkertainen, mutta silti ilmaisukykyinen ohjelmointikieli. Siitä on saatu positiivisia kokemuksia ainakin yläasteelaisten parissa, ja useissa suomalaisissa korkeakouluissa Python on ensimäinen opetuskieli. Sitä käytetään opetuksen lisäksi kaupallisissa sovelluksissa esimerkiksi Googlella. Kuitenkin, alaluokkalaisille Python-ohjelmointi voi olla kovin haastavaa, joten se voi olla sopivampi jo kerran Scratchiä harjoiteille tai yläasteen kynnyksellä oleville.

Python on yleisyytensä takia melko hyvin dokumentoitu. Varsinkin englanniksi materiaalia on runsaasti saatavilla, mutta myös suomeksi on hyviä kirjoja sekä nettioppaita.

Scratch-resursseja

- http://scratched.media.mit.edu/
- http://scratch.mit.edu/

Ohjelmoinnin opetuksesta yleisesti

- http://www.imped.fi/

Python-kielestä

- https://www.doria.fi/handle/10024/63381

Oppilaat olivat innostuneita ohjelmoinnista ja olisivat halunneet toteuttaa haastavampia ohjelmia kuin Scratchilla on mahdollista. Oppilaat tekivät tehtäviä eri tahtiin, mikä kannattaa huomioida kerhoa suunniteltaessa. Oppilaat myös sisäistivät asioita omaan tahtiinsa, toiset eivät tarvinneet kertausta viime viikon asioista, kun taas toiset kysyivät apua jokaisessa tehtävässä. Saattaakin olla hyödyllistä, jos kerhossa on paikalla useampia opettajia varsinkin jos ryhmän koko on kymmentä suurempi.

Ohjelmoinnissa ei aina ole yhtä ja oikeaa tapaa tehdä asioita, mikä näkyy myös oppilaiden tehtävien ratkaisuissa. Vaikka oppilaiden ratkaisut eivät olisikaan hyviä, kannattaa oppilaiden luovuutta tukea ja antaa heidän toteuttaa omia ideoitaan. Näin oppilaat rohkaistuvat tekemään ja kertomaan omista kotona tekemistään projekteista.

Lisäksi kaikkien ohjelmien kohdalla kannattaa keskustella, miksi toiset ratkaisut ovat parempia kuin toiset, sekä perustella miten tässä ongelmassa päädyttiin tähän ratkaisuun. Muista, että ohjelmoinnissa epäonnistuminenkin voi olla oppimiskokemus.

Uskomme myös, että ohjelmointikerhot parantavat ongelmanratkaisukykyä ja matemaattista päättelykykyä. Myös omat pienimuotoiset testimme sekä muu aihepiiriä sivuava tutkimus osoittavat vihjaavat tähän suuntaan. Tämä voi olla piilotavoite, jota ei tietenkään erikseen tarvitse mainita oppilaille, mutta sitä voidaan käyttää apuna kerhon toiminnan perustelussa.

Olemme joillain pitämillämme kerhoilla lopussa varsinaisen opetuksen sijaan pitäneet projektityöjakson. Tällöin idea on, että oppilaat itse ideoivat ja toteuttavat loppuun jonkun yksinkertaisen projektin - esimerkiksi pelin - meidän ollessa vain apuna vaikeiden osien suorituksessa.

Projektijakso on auttanut oppilaita toteuttamaan omia ideoitaan ja käyttämään tietotekniikkaa välineenä omien toiveiden toteuttamiseen. Lisäksi se on mahdollistanut hyvin kertaamisen opituista sisällöistä. Lopputyöt voi toki laittaa esille koko koululle esimerkiksi julisteiden muodossa.

Onnea ohjelmointikerhon kanssa,
Matti, Noora ja Nyyti sekä muut ohjelmointikerhon pitämiseen osallistuneet

Kuulemme mielellämme, miten materiaalia voisi kehittää. Lisäksi haluaisimme kuulla, mitä hyötyä tästä materiaalista on ollut koulun opetustyössä käytännössä.

Meidät tavoitat parhaiten sähköpostitse scratch-palaute@kerhokeskus.fi.

Sanasto

Costume kuvaa yksittäisen hahmon kulloista ulkonäkö.

For-rakenne Toistorakenne, jossa toistojen määrä on määritelty kokonaisluvulla. Kts. Toistorakenne.

Forever-rakenne Toistorakenne, jota toistetaan loputtomasti. Kts. Toistorakenne.

If-lause Rakenne, jossa if-lauseen sisällä oleva koodi suoritetaan vain, jos sille asetettu ehto on tosi. Kts. ehtolause

Ehtolause Rakenne, joka sisältää ehtoja tiettyjen koodinpätkien suorittamiseen.

Else Elsen sisällä oleva koodi suoritetaan vain, mikäli if-lauseen ehto ei toteutunut

Ja-operaattori Palauttaa toden, mikäli molemmat ehdot ovat tosia. Kts. Looginen operaattori

Kynä on Scratchin matalan tason piirtotoiminnallisuus.

Lista Rakenne, jonka avulla voit tallentaa monta muuttujaa samaan paikkaan. Kts. muuttuja

Looginen operaattori Yhdistää useita totuusarvollisia lausekkeita yhdeksi totuusarvoksi.

Modulo Jakojäännös. Esim 5/2 modulo on 1 (2*2=4 ja 5-4 = 1)

Muuttuja Tietty muistipaikka tietokoneen muistissa, johon viitataan muuttujan nimellä. Arvoa voi muuttaa.

Ohjelma Kokonaisuus, jonka koodi muodostaa.

Ohjelmointi Komentojen kirjoittaminen tietokoneelle, koodaus.

Ohjelmointikieli Formaalia, ihmiselle ymmärrettäväksi tehtyä konekieltä.

Parametri Tarkenne joka annetaan jollekin toiminnolle. Esim. Yhteenlaskussa 3 + 5, 3 ja 5 ovat yhteenlaskun parametreja.

Projekti Kokonaisuus, jossa ovat kaikki samaan asiaan liittyvät koodit.

Signaali Lähettää viestin. Kts. Tapahtumapohjainen ohjelmointi.

Skripti Lyhyt suoritettava koodinpätkä.

Sprite/hahmo Hahmo, jolle skriptejä voidaan luoda.

Slot ottaa vastaan viestin. Kts. Tapahtumapohjainen ohjelmointi.

Stage/Tausta Oikeassa ylänurkassa näkyvä laatikko, jossa skriptien suoritus näkyy.

Tai-operaattori Palauttaa toden, mikäli vähintään toinen ehdoista on tosi. Kts. Looginen operaattori

Tapahtumapohjainen ohjelmointi mahdollistaa hahmojen välisen kommunikoinnin.

Toistorakenne Rakenne, jolla toistetaan koodin pätkää. Toiston määrä/ehto riippuu toistorakenteesta.

Totuusarvo Kertoo lauseen paikkansa pitävyyden, tosi tai epätosi

While-rakenne Toistorakenne, joka toistaa niin kauan kun sille annettu ehto on tosi. Kts. Toistorakenne