ESIMERKKEJÄ KOODISTA

Tämän oppaan piirissä ei ole tarkoituksenmukaista tai edes mahdollista mennä siihen, miten varsinaista ohjelmakoodia kirjoitetaan. Eri ohjelmointikieliä on satoja, ja niiden oppimista varten löytyy lukemattomia oppaita ja verkkoresursseja.

Tämän osion tarkoituksena on kuitenkin vilauttaa maallikolle hiukan sitä, miltä kirjoitettu ohjelmakoodi esimerkiksi näyttää – ja miten eri kielet ja kielten tyypit karkeasti ottaen eroavat toisistaan esimerkiksi sen suhteen, millä tarkkuudella niillä operoidaan.

Seuraavat esimerkit havainnollistavat asiaa.

ESIMERKKI 0: VOILEIVÄN TEKEMINEN

Ohjelmoinnin tavoitteena on opettaa tietokonetta tekemään toistuvasti sellaisia asioita, joita ihminen ei halua tehdä joka kerta käsin erikseen. Viimeisen kahdenkymmenen vuoden aikana kieliä on kehitetty ja koodia on kirjoitettu ja niin paljon, että monet sellaiset tehtävät, jotka ennen veivät satoja tai jopa tuhansia rivejä koodia saa moderneissa kielissä ilmaistua muutamalla rivillä.

Palataan sivun 16 voileipäesimerkkiin:

Eri ohjelmointikielet vaativat eri tarkkuuden kuvailua siitä, mitä halutaan tehtäväksi.

Joissain kielissä tietokoneelle pitää selittää jokainen vaihe hyvin tarkasti, mutta lopputuloksena ohjelmoijalla on täydellinen vapaus luoda täysin sellaista jälkeä, kuin itse haluaa.

Toiset kielet lähtevät siitä, että ohjelmoijalle riittää hyvä perussuoritus. Sen vuoksi kielen kehittäjät ovat tehneet yleisiin ongelmiin tukun valmiita ratkaisuja, joita ohjelmoija voi käyttää hyväkseen. Tämä nopeuttaa koodin kirjoittamista ja ohjelmointiongelmien ratkaisemista.

Asiaa voisi verrata vaikkapa siihen, haluatko maustaa lihakeiton liemen kasvattamalla itse omat yrttisi ja mausteesi (yksityiskohtaisia ohjeita vaativat kielet) – vai riittääkö sinulle, että laitat keitetyn veden sekaan kaupasta ostetun lihaliemikuution (modernit helppokäyttöiset kielet).

Monille meistä jälkimmäinen riittää, mutta ammattikokki voi haluta täyden kontrollin.

Voileipäohjeiksi käännettynä muutama eritasoinen ohjelmointikieli voisi näyttää tältä ihmisen kielellä kirjoitettuna:

Valmiin toiminnallisuuden sisältävä kieli, esimerkiksi JavaScript tai Ruby

```
Tee voileipä.
```

Keskitason ohjeita vaativa kieli, esimerkiksi C

```
Kävele kaapille.
Ota kaapista leipäpussi.
Aseta leivät pussista pöydälle.
Voitele leipä.
```

Hyvin yksityiskohtaiset ohjeet vaativa kieli, esimerkiksi konekieli Assembly

```
Ota askelia kohtisuoraan, kunnes
saavut keittiön kaapille.
Kun saavut kädenmitan päähän
kaapista, pysähdy.
Nosta vasen kätesi.
Avaa kaapiston vasen ovi tasan 90
asteen kulmaan kaapin alareunaan
nähden.
Käyttäen oikeaa kättäsi tartu
ylimmällä hyllyllä olevaan
leipäpussiin.
Siirrä pussi pöydälle.
Jos pussissa on suljin, poista se.
Tartu pussissa olevaan ensimmäiseen
leipään.
Ota leipä pussista ulos.
Laske leipä pöydälle.
...
```

ESIMERKKI 1: HELLO WORLD!

Ohjelmointipiireissä on 1970-luvulta lähtien yleistynyt sympaattinen tapa: kun uutta kieltä lähdetään opettamaan, perinteisesti ensimmäinen asia on opetella, miten kielen avulla tulostetaan tietokoneen näytölle sanat ”Hello World!”.

Hello World! -fraasin tulostaminen tapahtuu useimmissa kielissä yksinkertaisilla koodiriveillä, jotta aloittelija voi opastettuna seurata ohjelman toimintaa ja ymmärtää ohjelmointikielen perusteita.

Alla on esitetty neljä esimerkkiä siitä, miten Hello World! -sanat tulostuvat näytölle eri ohjelmointikielillä.

Lukijan ei tässä tarvitse ymmärtää, miten kielet tarkemmin ottaen toimivat. Esimerkkien tarkoitus on esitellä, miten erilaiselta sama tehtävä voi näyttää eri kielillä.

Huomaa, että alla läpikäydyt neljä ohjelmointikieltä ovat samat kuin yllä mainitussa voileipäesimerkissä ja etenevät yksinkertaisesta monimutkaisempaan suuntaan.

Ruby – ei näytä kovin vaikealta

```
puts "Hello World!"
```

JavaScript – mukiinmenevää tämäkin

```
console.log("Hello World!")
```

C-kieli – vaatii jo perehtymistä

```
#include <stdlib.h>
#include <stdio.h>

int main(void)
{
 printf("Hello World!\n");
 return EXIT_SUCCESS;
}
```

8086 Assembly – monimutkaista (eikä juuri enää käytössä)

```
DOSSEG
.MODEL TINY
.DATA
TXT DB "Hello World!$"
.CODE
START:
       MOV ax, @DATA
       MOV ds, ax

       MOV ah, 09h          ; prepare output function
       MOV dx, OFFSET TXT   ; set offset
       INT 21h              ; output string TXT

       MOV AX, 4C00h        ; go back to DOS
       INT 21h
END START
```


ESIMERKKI 2: NUMERONARVAUSPELI

Eri ohjelmointikieliä kirjoitetaan eri merkinnöin, ja samoja asioita ilmaistaan hiukan eri näkökulmista.

Alla olevalla koodilla saa luotua pienen pelin, jossa tietokone arpoo ensin jonkin "ajattelemansa" numeron, jonka jälkeen käyttäjä koettaa arvata tuon numeron.

Monissa ohjelmointikielissä on esimerkiksi valmiiksi kirjoitettuja toimintoja (näitä kutsutaan funktioiksi), jotka valitsevat automaattisesti satunnaisen numeron ja muuttavat sen kokonaisluvuksi. Jotta ohjelmointia tuntematon lukija voisi seurata alla olevia esimerkkejä edes päällisin puolin, koodipätkiä on selitetty auki tavallisella kielellä – käytäntö, jota myös koodarit harrastavat!

Älä missään nimessä ahdistu, jos alla oleva näyttää vaikealta. Ohjelmointia opetellaan askeleittain, ja vaikka alla olevat esimerkit ovat suhteellisen yksinkertaisia, niitä ei käytännössä voi täysin ymmärtää ilman kyseisen kielten perusteiden opettelua.

Coffee Script

```
num = Math.ceil(Math.random() * 10)
guess = prompt "Arvaa mitä lukua välillä 1-10 ajattelen!"
while parseInt(guess) isnt num
 guess = prompt "Hävisit! Arvaa uudestaan."
alert "Oikein arvattu!"
```

Aluksi määritellään funktio nimeltä "num". Se valitsee satunnaisen luvun väliltä 1–10. Apuna tässä käy-

tetään CoffeeScript-kielen valmiita työkaluja (”Math.ceil”- ja ”Math.random”-funktiot). Valittu luku on siis pelin ”oikea vastaus”, joka käyttäjän pitää arvata.

Seuraavaksi määritellään ”guess”-niminen funktio. Se näyttää käyttäjän näytöllä viesti-ikkunan, joka tiedustelee käyttäjää arvaamaan, mitä numeroa välillä 1–10 tietokone ”ajattelee”.

Sitten todetaan, että jos käyttäjän syöttämä arvattu numero ei vastaa ensimmäisen rivin satunnaisesti valittua, tietokoneen ”ajattelemaa” numeroa, näytetään käyttäjälle rivillä neljä oleva viesti. Sen mukaan käyttäjä ei arvannut oikein.

Jos käyttäjä arvaa oikein – eli syötetty luku on sama kuin ensimmäisellä rivillä oleva num – näytetään käyttäjälle viesti, jonka mukaan hän on arvannut oikein.

PYTHON

```
import random

target, guess = random.randint(1, 10), 0
while target != guess:
guess = int(input('Arvaa ajattelemani luku välillä 1–10 kunnes saat sen
oikein: '))
   print('Oikein arvattu!')
```

Tässä ”random.randint” on Python-kieleen sisäänrakennettu funktio, joka valitsee sattumanvaraisen numeron yhden ja kymmenen välillä. Nolla lopussa tekee luvusta kokonaisluvun (kuten sanottua, älä huoli, jos esimerkit eivät tunnu ymmärrettäviltä ilman perehtymistä kielen logiikkaan!).

Koodissa todetaan, että niin kauan kuin käyttäjän arvaus ”guess” ja tietokoneen ajattelema luku ”target” eivät ole yhtäsuuria, käyttäjää pyydetään arvaamaan yhä uutta numeroa.

Seuraavaksi esitetään viesti-ikkuna, joka pyytää käyttäjää syöttämään luvun välillä 1–10, kunnes hän arvaa tietokoneen ”ajatuksen” oikein.

Kun ”guess” ja ”target” ovat lopulta samat, tietokone kertoo, että käyttäjä on arvannut oikein.

”DIGITALISAATIO EI TARKOITA, ETTÄ KÄDENTAIDOT VÄHENEVÄT”

Tapio Tuomi, *asiantuntija, Sitra*

”Kun maailma muuttuu, ja jos Suomi haluaa pärjätä, niin digitalisaatio on homman juoni. Siihen tarvitaan erilaisia taitoja kuin nykyään. Paras tapa varmistaa menestys on se, että uusia taitoja opetetaan peruskoulussa kaikille suomalaisille.

Kyse on tasa-arvosta. Kun ohjelmointi laitetaan peruskouluun, se antaa kaikille samat mahdollisuudet jatkaa eteenpäin.

Toinen tärkeä asia on kädentaitojen uudistaminen. Villamyssyä voidaan aina virkata käsityötunnilla, mutta myös uusia taitoja täytyy olla. Koodatessa tekee itse jotain, ja syntyy jotain uutta.

Minä jauhan esimerkiksi itse kotona jyvistä jauhoni. Ymmärrys siitä, mistä asiat rakentuvat, on ohjelmoinninkin osalta hyödyllistä.

Digitalisaatio ei tarkoita sitä, että kädentaidot vähenevät, ne vaan muuttavat muotoaan. Ohjelmointikyky on työkalu, jonka avulla rakennetaan asioita. Voidaan myös ajatella niin, että halutaan pitää huolta Suomen ja suomalaisen koulusysteemin maineesta.

Muut maat saavat koko ajan Suomea kiinni ja menevät ohi. On hyvä päivittää ja uusia koulun sisältöjä – kuten nyt vaikka sitä kautta, että ilmeisesti nyt jakokulma on jäämässä matematiikasta pois.

Peruskoulutusosastolla tajutaan itse muuttaa asioita oikeaan suuntaan, se on minusta hienoa.

Pysähtyminen pitkäksi aikaa on pahasta. Tärkeämpää on mennä johonkin suuntaan. Liian valmiiksi sitä ei voi suunnitella, muuten ei tehtäisi yhtään mitään. Sitten jos tulee vastaan ongelmia, ratkaistaan ne sitten.”