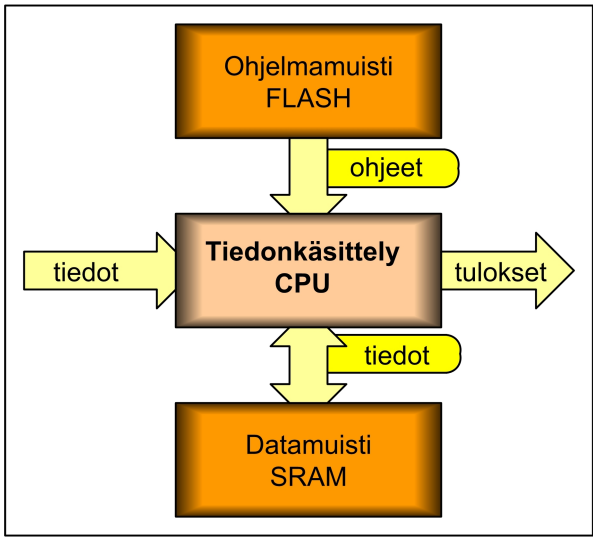
Tietokoneohjelman anatomia
Mikä on tietokoneohjelma ja tietokonekieli

PUNOMO NETWORKS OY
28.9.2017 pva

https://punomo.fi/?p=52266&preview=true

”Nothing is particularly hard if you devide it into small jobs.”
- Henry Ford

Tietokone tunnetaan, sehän on näyttö, hiiri ja näppäimistö, mutta mikä on tietokoneohjelma? Tietokone ei tiedä yhtään mitään, eikä se pysty ajattelemaan samalla lailla kuin ihminen. Ihminen osaa tulkita hyvinkin monimutkaisia asioita tai päätellä kiroitusvihreestä huolimatta vaikkapa kirjeensaajan oikean nimen ja osoitteen. Tietokone ei tähän pysty. Se pystyy tulkitsemaan vain numeroita ja niistäkin vain ykkösiä ja nollia. Tästä johtuen tietokoneen ns. tietojen käsittely on vain hyvin yksinkertaista laskentaa ja loogista päättelyä. Mutta kun se tekee sitä niin nopesti. Siinä on se juju.



Tiedon käsittelyn idea

Käsiteltävä tieto tuodaan tietokoneeseen joka näppäimistöltä, ulkoisilta laitteilta tai tietoverkosta.
Tieto työstetään tietokoneen mikroprosessorissa, välitulokset talletetaan datamuistiin.
Tieto työstetään ohjelmamuistista saatujen ohjelman ohjeiden (käskyjen) mukaan.
Tiedon työstämisen tulokset viedään näyttöruudulle, tulostetaan paperille, talletetaan ulkoiselle laitteelle, työnnetään tietoverkon kautta palvelintietokoneelle tai pannaan ohjaamaan jotain ulkoista laitetta.

Mikä on tietokoneohjelma?
Tietokoneohjelma on toimintaohje tietokoneelle. Kone tekee kaiken sen - ja vain sen - mitä ohjelmoija ohjelmassa määrittää. Ohjelma koostuu peräkkäisistä komennoista eli käskyistä. Käskyt on tallennettu ohjelmamuistiin, josta ne haetaan yksi kerrallaan ja viedään mikroprosessorin (CPU) käskynkäsittely-yksikköön, jossa ne tulkitaan ja tehdään käskyn määräämä pieni, hyvin yksinkertainen toiminto. Riittävän paljon erilaisia pieniä toimintoja, hyvin, hyvin nopeaan tahtiin peräjälkeen suoritettuna muodostaa ajettavan ohjelman.

Ohjelma:
- sellaisia peräkkäisiä binäärilukuja (ykkösiä ja nollia) ohjelmamuistissa, jotka CPU osaa tulkita transistorien ohjausjännitteiksi.

Tietokone tekee kaikkea kivaa ohjelman ohjaamana. Tästä seuraa, että tulevaisuudessa tarvitaan lisää ohjelmoinnin osaajia. Minimi on se, että jokaisen kansalaisen on ymmärrettävä, ainakin jollain tasolla, tuon tietokoneistetun yhteiskunnan toiminta.
Tietokoneohjelmia kirjoitetaan ns tietokonekielillä, joita on monenlaisia.

Mitäs sitten on ohjelmointi, entä ohjelmointikieli?
Miten niitä opitaan?
Tarkoitus on saada tietokone tekemään asioita, joista on hyötyä käyttäjälle. Kyseessä voi olla tekstin oikoluku, lehtien tai sähköpostin selaaminen tai pelien pelaaminen tai kodin tekniikan ohjaus ja valvonta, teollisuuden koneiden ja laitteiden ohjaus ja valvonta, tms. Jokainen näistä on vaatinut ohjelmointia.

Ohjelmointi
Kun ohjelmointia opetetaan/opitaan, se on tehtävä ympäristössä jossa oppilaalle syntyy halu oppia. Siksi ohjelmointiin on käytettävä oikeita ohjelmointikieliä, ei mitään leikkimistä. Ja tottakai työkaluohjelmien on oltava helppokäyttöisiä ja tehokkaita. Koodattavan ohjelman tulee heti alkutoimien jälkeen (muutama oppitunti?) tehdä jotain mielenkiintoista, näkyvää, kuten vilkuttaa LED-lamppua, soittaa summeria tai pyörittää pientä sähkömoottoria. Myöhemmin laite-internettiä. Ensimmäisellä tunnilla toki koodataan perinteinen ”Hello World”.

Ohjelmaa suoritetaan käskyrivi kerrallaan ja muuttujiin (muistipaikkoihin) sijoitellaan arvoja, arvoilla suoritetaan laskentaa, tulosten perusteella tehdään vertailuja, valintoja, ohjataan jotain tai tehdään jotain muuta tärkeää. Nämä toimet ovat periaatteessa samoja kaikessa ohjelmoinnissa, eli näin opitut taidot ovat yleishyödyllisiä. Nämä taidot voi opetella melkein millä kielellä tahansa. Järkevintä on aloittaa ohjelmoinnin opiskelu helpoimmasta päästä.

Ylläolevasta seuraa se, että ohjelmoinnin opetus pitää järjestää perusopintojen jälkeen laiteläheisesti. Opittavalla koodilla ohjataan ja valvotaan tietokoneeseen liitettyjä antureita ja toimilaitteita. Näin opitaan ajattelemaan käytännön läheisesti ja älykkyyttä sisältävien laitteiden toiminnan hahmottaminen ja ymmärtäminen on helpompaa.

Ohjelmointi ei välttämättä ole monimutkaista. Ongelmat, joita tietotekniikalla ratkaistaan, ne ovat monimutkaisia. Ohjelmointityö on loogista hommaa ja työkalut (ohjelmointikielet ja koodi-editorit) selväpiirteisiä ja kohtuudella haltuunotettavissa. Silti jokaisesta ei tule huippukoodaajaa, kuten ei tule formulakuskiakaan. Eikä tarvitsekaan.

Vaikka oikea ohjelmointityö on ammattilaisten työtä, koodaaminen voisi olla keskivertokansalaisten älykäs harrastus. Tietotekniikan ja ohjelmoinnin perusteiden osaaminen laajentaa jokaisen tämän päivän yhteiskunnan jäsenen ajattelemaan ”tietokoneellisesti”. Ja sillä osaamisella pärjää kiivaasti teknistyvässä arjessa. Vaikka tulisi vastaan minkälaista laitteiden internettiä.

“Tietokoneajattelu” auttaa meitä hyödyntämään nykytelevision kaikkia ominaisuuksia, käyttämään tietokonetta muuhunkin kuin pelaamiseen, puhelinta muuhunkin kuin soittamiseen, hyödyntämään internettiä vaikkapa päivällisen valmistuksessa tai kodin sisustuksessa, viestittämään Skypellä Pihtiputaan mummolle terveiset, jne. Vaikkei itse ajaisi autoa, on tunnettava liikennesääntöjä ja -kulttuuria sen verran ettei jää autojen alle.

Mille ohjelmointitaito rakennetaan
Ohjelmoijaa auttaa hyvä kielitaito, pohjana laadukas äidinkielen taito. Parhailla ohjelmoijilla on hyvin laaja yleissivistys, ynnä monenlaista tietotaitoa eri elämänalueilta. Looginen päättelykyky on peruskivi jolla pääsee pitkälle.
Ohjelmoijan tulee tuntea ohjelmointikielen mahdollisuudet ja tietoa ja taitoa siitä aiheesta, johon koodia kirjoitetaan. Näin on erikoisesti laiteläheisessä, IoT (Internet of Things) ohjelmoinnissa. Mitä se sitten aikanaan tulee olemaankaan.
Tärkeää on ymmärtää heti aluksi, että ohjelmoinnin oppiminen on eri asia, kuin ohjelmointikielen oppiminen. Ohjelmointitaito ei riipu kielestä. Se on taito ymmärtää miten yksinkertaisia, mutta selkeästi määriteltyjä käskyjä yhdistämällä tehdään (joskus hyvinkin) monimutkaisia asioita.
Ohjelmointitaito kehittyy ahkeruuden, kokemuksen ja kokeneitten myötävaikutuksella.
Ohjelmointikielen oppiminen on helpompaa.

Mikä on ohjelmointikieli?

Opettaja:
Ohjelmointikieli on ohjelmien esittämiseen tarkoitettu, prosessorikohtaiseen konekieleen käännettävissä oleva kieli.

Tietosanakirja:
Ohjelmointikieli on joukko käsitteitä, joilla ohjelmoija määrittelee algoritmin rajaamat suoritettavat tehtävät.

Ohjelmoija:
Ohjelmointikieli on työkalu, jonka avulla ohjelmoija miettii mitä on mahdollista tehdä ja mitä ei.

Ohjelmointikieli on väline jolla ihminen kertoo tietokoneelle, mitä sen pitää tehdä. Ensin se kuitenkin pitää kääntää tietokoneen ”ymmärtämään” muotoon.
Mitä ohjelmointikieltä tulisi oppia?
Sitä ei tiedä kukaan.

Erilaisia ohjelmointikieliä

Konekieli
Tietokoneen mikroprosessori (CPU) käsittelee tietoja ohjelman määräysten mukaisesti. Ohjelma, siis sarja peräkkäisiä käskyjä, koostuu biteistä eli ykkösistä ja nollista.

Prosessoritasolla:
1 = jännite on, virta kulkee, kytkin kiinni
0 = jännitettä ei ole, virta ei kulje, kytkin auki

Jokaisen tietokoneen CPU “ymmärtää” vain sille koodattuja bittikuvioita. Intelin suoritin “ymmärtää” vain Intelin konekoodia, RasPin ARM omaansa, jne.
Tietokoneen ymmärtämiä bittijonoja, erilaisia ykkösten ja nollien yhdistelmiä, nimitetään konekieleksi, machine language. Ne ovat joko 8/16/32/64-bittiä pitkiä, esim näin:
1011 1100
0101 1010
1101 1110

Konekielisissä ohjelmissa on kahdenlaisia osia: operaatioita eli käskyjä ja operandeja eli numerotietoa. Ohjelman käskyt ja käsiteltävä tieto, data, talletetaan muistiin, jossa jokaiselle bitille on oma lokero. Operaatioiden avulla ohjelmissa manipuloidaan muistipaikkojen sisältöjä. Tietokoneen osaamat toimenpiteet eli operaatiot ovat hyvin yksinkertaisia. Tietokone osaa esimerkiksi laskea yhteen kaksi lukua ja tarkistaa, onko tulos nolla. Koska tietokoneessa kaikki asiat esitetään bittien avulla, konekieliset ohjelmat ja niiden käsittelemä data ovat käytännössä peräkkäistä ykkösten ja nollien jonoa. Tuon jonon käsittely on ihmiselle hankalaa.

Symbolinen konekieli, eli assembly

Ohjelmoinnin helpottamiseksi kehitettiin symbolinen konekieli, assembly, asm. Siinä käskyjä ei esitetä enää bittijonoina, vaan jokaista käskyä varten on sovittu määrätty apusana, muistikas, mnemonic. Esimerkiksi yhteenlaskukäskyn muistikas on add ja vähennyslaskukäskyn sub. Niitä on helpompi hallita kuin pelkkiä ykkösnollia. Käskyt sisältävät myös tiedon siitä, mistä käsiteltävät luvut löytyvät. Assembly-kielellä kirjoitettu ohjelma käännetään CPU:n ymmärtämälle konekielelle assembler-käännösohjelmalla. Assembly-ohjelma on aina prosessorikohtainen. Koska asm on laiteläheinen, koodi kohdistuu suoraan prosessorin rekistereihin ja siksi niitä tarvitaan paljon pienenkin asian koodaamiseen. Siten se vaatii ohjelmoijalta osaamisen lisäksi kärsivällisyyttä ja ennenkaikkea aikaa. Ohjelman tekeminen asmilla on todella suuritöistä, mutta toisaalta siten syntyy nopeaa koodia.

Lausekielet

Koska symbolinen konekieli on hankala käyttää, tarvittiin tilalle jotain helpompaa. Aloitettiin rakentamaan lausekieliä. Lausekielisissä ohjelmissa monta peräkkäistä konekielen käskyä on korvattu yhdellä lausekielen käskyllä. Käskyt on suunniteltu niin, että ne vastaavat ohjelmoijan ajattelua ja (englannin) kieltä.

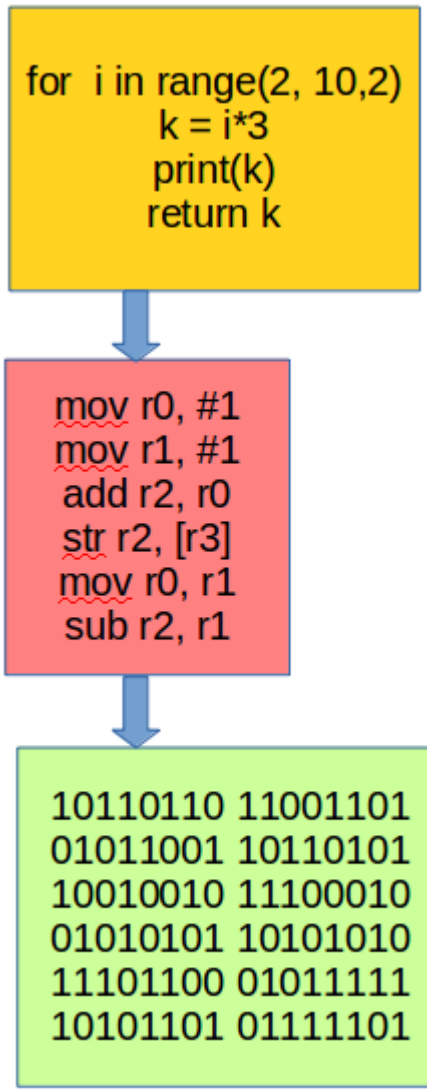
Lausekielet eli korkean tason kielet (high level language) muistuttavat puhuttua kieltä. Ne ovat “lähellä ihmistä”. Sanavarasto on suppea ja niitten käyttöä ohjaa ja määrittää tarkat kielioppisäännöt eli syntaksi (syntax). Kaikilla asioilla on lisäksi hyvin tarkat merkitykset eli semantiikka (semantics).

Lausekielet ovat laiteriippumattomia, eli ohjelmointi ei riipu siitä tietokoneesta, jolla ohjelma ajetaan. Lausekieli eli lähdekoodi on käännettävä tai ajon aikana tulkattava ko. tietokoneen prosessorin ymmärtämäksi konekoodiksi. Käännöksen suorittaa kääntäjä-ohjelma (compiler), tai ohjelman ajon aikana tulkki-ohjelma (interpreter).

Kääntäjä kääntää koko lausekielisen ohjelman kerralla, muuttaa sen konekielelle ja tallentaa ajokelpoiseksi tiedostoksi. Se on hyvin monimutkainen prosessi. Selvitän toisaalla. Konekielinen ohjelma voidaan ajaa milloin ja miten monta kertaa tahansa. Näin menetellään esim C-kielisen koodin kanssa.

Tulkki tulkitsee lausekielistä ohjelmaa käsky kerrallaan, muuttaa käskyn konekielelle ja suorittaa käskyn välittömästi. Tämän jälkeen tulkki siirtyy lausekielisen ohjelman seuraavalle riville ja seuraavaan käskyyn. Tulkattuja käskyjä ei tallenneta minnekään, joten jos ohjelma halutaan ajaa uudelleen, pitää se myös tulkita uudelleen. Näin menetellään esim Python-koodin kanssa.

Ihmisen ja tietokoneen kommunikointihierarkia ohjelmoinnin kannalta.



High level code, Python

Assembly code

Binary code

Ohjelmointikielien hiearkia.

Ohjelmoija, joka osaa ohjelmointikielten yleiset periaatteet (joita opiskelet parast'aikaa), pystyy ja haluaa opiskella useita eri kieliä.
Siten hän voi valita kuhunkin tarkoitukseen sopivimman kielen.

Ohjelmointikielen valintakriteerejä
Tietokoneohjelmia on monenlaisia, samoin ohjelmointikieliä joilla ohjeet koneille kirjoitetaan. Ohjelmointikieli on oikeastaan vain työkalu jolla ihminen pystyy kertomaan tietokoneelle, mitä sen milloinkin pitää tehdä. Ohjelmoijan tulee tuntea kielen ominaisuudet eli mahdollisuudet ja sen lisäksi se ympäristö, jossa ohjelmaa ajetaan. Kirjanpito-ohjelman tekijän tulee tuntea kirjanpito-käytännöt ja lainsäädäntö, kun taas raudanläheisen kännykkäohjelman tekijä kohtaa aivan erilaiset ongelmat. Ohjelmointikielen oppiminen ei ole vaikeaa, työlästä kylläkin, mutta toisaalta erittäin mielenkiintoista. Kunhan opiskellaan oikeita asioita ja oikealla tavalla.

Kieliä on kehitelty vuosien varrella tusinoittain periaatteella: joka työhön oma kieli.
Opiskeltavan ensimmäisen kielen valinnasta on kiistelty yhtä kauan. On "välttämätön ensin"-topologiaa, oliopohjaista ohjelmointia, toiminnallisuus- ja algoritmilähtöistä ideologiaa ja ties vaikka mitä.
Meillä on selvä ajatus, joka on ollut selvä jo ennenkuin kukaan oli kuullutkaan hittisanaa IoT, Internet of Things. Ihmiset ovat jo netissä, nyt sinne ovat menossa laitteet. Se edellyttää laiteläheistä ohjelmointia. Siksi haluamme tehdä laiteläheistä koodia. IoT edellyttää sitä.

Ensimmäisen kielen, sen jolla opiskellaan ohjelmointia, on oltava **"lähellä ohjelmoijaa"**:
* ohjelmoijan kannalta sen tulee olla ikään kuin oikea ihmisen kommunikaatioonsa käyttämä kieli, koska se siten on helposti otettavissa haltuun.

Sen jälkeen siirrytään **laiteläheiseen ohjelmointiin**, joten kielen on oltava **"lähellä konetta"**:
* siten, että CPU:n rekistereitä ja oheispiirejä (liitäntäpiirejä) voidaan manipuloida bittitasolla.

Lisää syitä.
* Maksimi hyöty - minimi työ eli vähillä perustiedoilla pääsee tekemään toimivia ohjelmia.
* Samaa koodia tulisi voida ajaa eri laitteilla ja eri käyttöjärjestelmillä, kielen tulee olla ajoympäristöstä riippumaton.
* Tärkeä kriteeri: ohjelmointityökalujen tulee olla ilmaisia, löytyä netistä, tukena aktiivinen yhteisöä ja niillä tulee olla laaja käyttäjäkunta myös ammattilaisten keskuudessa, se takaa niiden jatkuvuuden.
* Työkalujen oltava laajennettavissa periaatteella sama työkalu useaan eri työhön. NetBeansin avulla opitaan ensin esim C-kieltä, myöhemmin asennetaan siihen lisämoduuli JavaScriptin opiskelua varten ja sen jälkeen HTML-moduuli.

Ei ole yhtä ainoaa oikeaa kieltä. Aloituskieli ei voi olla "oikea" tai "väärä". Kun ammattilainen ja osaava harrastelijakin käyttää useaa eri kieltä, niin aloituskieleksi tulisi valita se, joka on helpoin oppia.

Sitten vielä yksi kieli.
Mutta sitä ennen käsite algoritmi.

> Algoritmin idean oppiminen ja hyödyntäminen lienee vaikein osa ohjelmoinnin oppimista. Aiheesta on kirjoitettu useita kirjoja. Niihin kannattaa tutustua. Mutta vasta myöhemmin, kun ohjelmoijalla on tarpeeksi pohjatietoa asian omaksumiseen. Uusi tieto tarvitsee tartuntapintaa. Sulautettuihin systeemeihin ensiaskeleitaan ottavan tulee hallita ainakin algoritmien perusidea.

Algoritmi
Ohjelma käsittelee tietorakenteita, joista kohtaa lisää. Ohjekokoelma jonka mukaan noita tietorakenteita käsitellään, kutsutaan algoritmiksi (algorithm).
Algoritmi on kuvaus jonkin tehtävän suorittamiseksi tarvittavista toimenpiteistä, eli miten lähtötiedoista syntyy uutta tietoa, tulosteita. Algoritmi vastaa kysymykseen, ’mitä pitää tehdä ja missä järjestyksessä, jotta tavoite saavutetaan?’ Algoritmilla kuvataan ongelman ratkaisu niin yksinkertaisina loogisina askeleina, että sen perusteella voidaan kirjoittaa ohjelma.

Algoritmin eli ongelman ratkaisun voi kuvata puheena, graafisesti kuvilla (vuokaavio) tai pseudokielellä (lopullista ohjelmakoodia jäljittelevää tekstiä). Algoritmin käsite on laaja, eikä sillä tarkoiteta pelkkiä tietokoneeseen tai matematiikkaan liittyviä menetelmiä. Algoritmeja ovat kaikki täsmälliset suoritusohjeet, esim. keittokirja, virkkausohjekirja tai vaikka drinkkien sekoitusohje-kirja. Niiden ohjeita noudattamalla ruuan laiton, virkkaamisen tai drinkkien sekoittelun tulisi onnistua. Keittokirjan resepti on algoritmi ja raaka-aineet tietorakenteita.

Algoritmien perusvaatimukset
• Yleisyys: algoritmin on sovelluttava kaikkiin mahdollisiin määritellyn tehtävän eri tapauksiin
• Deterministisyys: tehtävän ratkaisun on oltava yksikäsitteisesti määritelty, niin että joka vaiheessa tiedetään täsmällisesti mitä seuraavaksi tehdään. Determinismi, lainalaisuusoppi, on filosofian näkökanta, jonka mukaan kaikilla asioilla on syynsä. Tästä seuraa, että kaikki, mikä tapahtuu, tapahtuu välttämättä, eikä mikään voisi tapahtua toisin.
• Tuloksellisuus: algoritmin on annettava aina oikea tulos, lisäksi sen suorituksen tulee aina päättyä.

Algoritmien laatiminen
• Asteittain tarkentuva menetelmä, jossa suunnitellaan ensin isot kokonaisuudet
ja sitten vaiheittain tarkennetaan pienempiin osiin, top-down design:

– ohjelman määrittely
    • toiminnot
        – funktiot
            » lauseet
            » lausekkeet
            » muuttujat
            » jne.

Algoritmista kannattaa ensin tehdä sanallinen kuvaus ja vasta myöhemmin lausemuotoinen kuvaus. Lausekielisesti esitetty algoritmi on jo lähellä ns. puoliohjelmointia.

Ja nyt siihen lisäkieleen
Parasta opetella käyttämään yhtä algoritmin kirjoittamismallia ja kunnolla. Tällöin valinta-ehdotukseni on pseudokieli, koska siinä tehtävän jäsentelyn ja ajattelun ohessa syntyy ohjelmalauseiden alkioita. Toisaalta pseudokoodin kirjoittaminen on saanut yhä lisääntyvää suosioita ohjelmoijien keskuudessa. Kaverille kans.

Pseudokieli
Pseudokieli on selväkielistä tekstiä, siis ihmiselle tarkoitettu kuvaus mitä ollaan tekemässä. Se on sekoitus suomea, englantia ja ohjelmointikielen termejä, joilla kuvataan tehtävän suoritusaskeleet. Jokaisesta moduulista (funktio) kirjoitetaan oma pseudokoodi. Pseudokoodin kirjoittaminen aloitetaan vasta sitten, kun ensin on selvitetty mikä on ongelma, mitä ollaan tekemässä, mikä on toimintaympäristö, mitä se edellyttää ohjelmalta.

Kun algoritmin kirjoittaa pseudokoodiksi, se on lähimpänä lausekielistä ohjelmarakennetta ja siten jatko on vähätöisempää. Tärkeää on jakaa monimutkainen tehtävä osatehtäviksi. Jos on tarvetta, nämä jaetaan edelleen pienemmiksi osatehtäviksi, kunnes päädytään helposti ratkaistavaan 'pikku ongelmaan'. Kun pseudokielen avulla on luotu kuvaus tehtävästä, niin sitten sen kääntäminen ja kirjoittaminen varsinaiseksi lähdekoodiksi on helppoa. Tai ainakin se on vähemmän vaikeaa. Algoritmi kirjoitetaan ihmisen ajattelua varten, mutta koko ajan miettien, miten työ on tietokoneella tehtävissä. Hyvin tehdystä algoritmista on helppoa kirjoittaa itse ohje, eli ohjelma tietokoneelle.

Ja sitten takaisin ohjelman rakenteen analysointiin.

Selvitän seuraavaksi ohjelmointiin ja ohjelmaan liittyviä käsitteitä jotta pääset helpommin alkuun. Käsitteiden syvällisempi oppiminen tapahtuu kurssin edetessä ja taitojen kasvaessa.
Esiteltävät asiat ovat ohjelmoinnissa yleiskäyttöisiä, ne esiintyvät kaikissa ohjelmointikielissä. Siten näiden perusasioiden syvällinen oppiminen helpottaa suuresti jatko-opintoja.

OHJELMAN PERUSELEMENTIT

Tunnus, identifier
Tunnukset ovat nimiä erilaisille asioille, jollaisia ovat mm. muuttujat, vakiot ja tyypit.

Muuttuja, variable
Muuttujaa käytetään tietovarastona erilaisille asioille. Muuttuja on bittien muodostama lokero tietokoneen muistissa. Muuttujan arvo on muistialueen sisältö. Muuttujan nimi viittaa muuttujan sisältämään tietoon. Muuttujista on perusteellinen selvitys Python-kurssin alkuharjoituksissa.

Tyyppi, type
Ohjelman tulee tietää onko joku-nimisessä muistipaikassa jokin luku, kirjain, teksti (merkkijono) vai jotain muuta, jotta ohjelmassa osataan tehdä muuttujalle oikeita toimia eli operaatioita.
Muuttujan tyyppi (type) kertoo ohjelmalle mitä muuttujalle voi tehdä.
Näitä ovat mm seuraavat:
int kokonaisluku (16 bittiä)
long kokonaisluku (32 bittiä)
float liukuluku
double kaksoistarkkuuden liukuluku
char merkki

Vakio, constant
Kaikissa ohjelmissa käytetään laskutoimituksissa yleensä joitain kiinteitä lukuarvoja.

Lauseet, statement
Ohjelman varsinainen toiminta tapahtuu lauseissa. Lauseessa tehdään jokin toimenpide, esimerkiksi kahden luvun yhteenlasku.

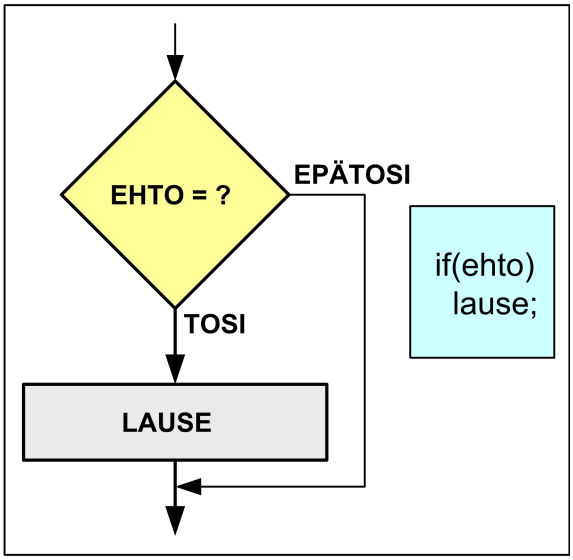
Kommentit, comments
Ohjelmiin voi ja niihin pitää liittää ohjelman toimintaa selittäviä tekstejä eli kommentteja. Kääntäjä/tulkki ei niitä noteeraa.

Rakenteinen ohjelmointi
Kun ohjelma koostetaan selkeistä ohjelmarakenteista puhutaan rakenteisesta ohjelmoinnista. Ohjausrakenteilla ohjataan ja muutetaan ohjelman kulkua eli käskyjen suoritusjärjestystä. Useimmat suorituksen ohjauskäskyt perustuvat jonkin ehdon testaamiseen. Ehto joko täyttyy, eli on tosi - true, tai ei täyty, eli on epätosi- false.
Jos tuntee ohjausrakenteet, silloin ohjelman toiminta on helppo ymmärtää ja tarvittaessa korjata tai muuttaa.

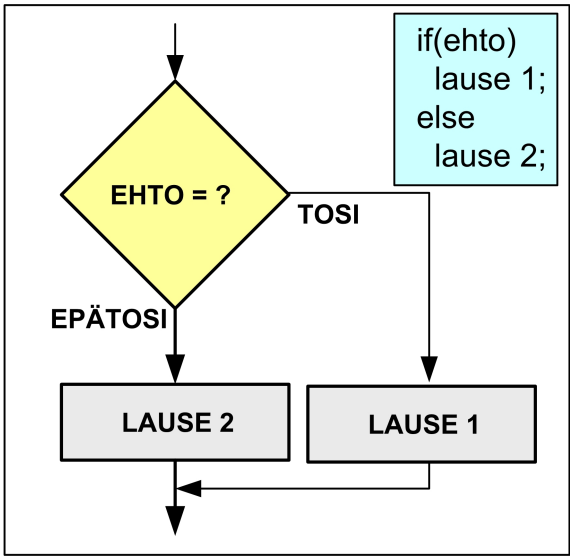
Ohjelman perusrakenteet
1. peräkkäisrakenne
2. valintarakenne
3. toistorakenne

1. Peräkkäisrakenne
ei selittelyä kaipaa, sarja toimenpiteitä peräkkäin. Ohjelma etenee suoraviivaisesti siten miten käskyt on ohjelmaan kirjoitettu.

2. Valintarakenne
tehdään jotain jonkun totta olevan ehdon perusteella. Jos if-ehto on tosi, niin tehdään lause tai lauseita ja ellei se ole tosi (on siis epätosi) niin sitten hypätään noiden lauseiden ohi ja ohjelma jatkuu. Nuolet kuvaavat ohjelman etenemistä.



If-else-rakenne
Joskus on tarpeen tehdä jotain siinäkin tapauksessa että ehto ei ole tosi. Tällöin valitaan vaihtoehto if-else rakenteella. Else suoritetaan siinä tapauksessa että ehto ei ole tosi.



Vuokaavioesitys if-else-rakenteelle

Ehdon ollessa tosi haaraudutaan oikealle ja suoritetaan lause 1, epätodessa tilanteessa suoritetaan lause 2. Molemmissa vaihtoehoissa jatketaan ohjelmaa if-else-rakennetta seuraavasta lauseesta.

Valinta useasta vaihtoehdosta
If-else lauseita voi olla useampiakin kuin seuraavan mallin mukaan.

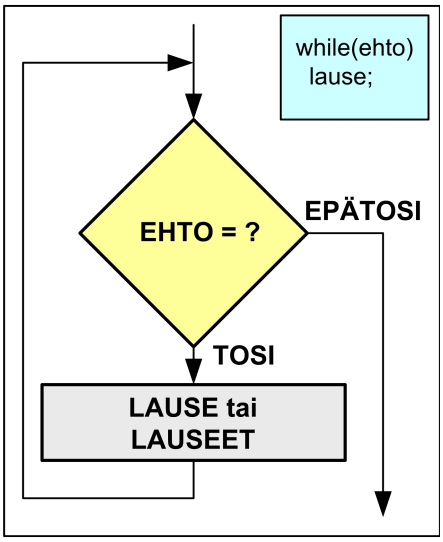
```
if (ehto)
  lauseet
else if
  lauseet
else
  lauseet
```

3. Toistorakenne,
samanlaisina toistuvien yksittäistapahtumien toistamista.
Toistorakenteet voivat olla alkuehtoisia tai loppuehtoisia.

Alkuehtoista ei välttämättä suoriteta kertaakaan.
Esimerkkinä alkuehtoisesta toistosta:
haluat keittää kahvit, tarkista onko kaapissa kahvia,
jos on, keitä kahvit,
ellei,
niin sitten et keitä.
Kahvit jää juomatta.

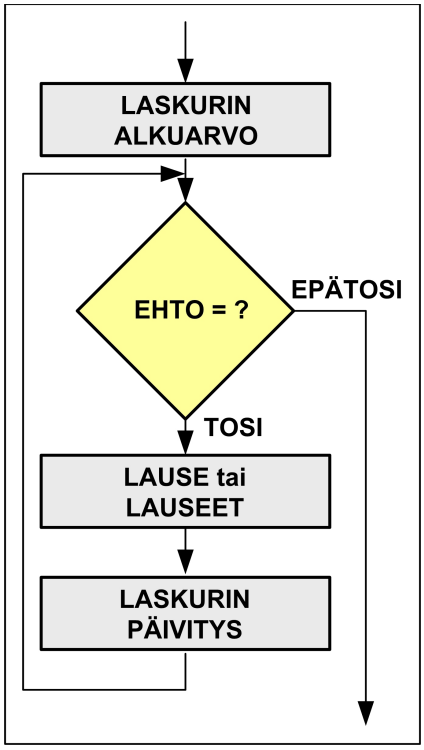
Loppuehtoinen suoritetaan aina vähintään yhden kerran.
Esimerkki loppuehtoisesta toistosta:
Haluat lainata kirjastosta viimeisimmän Linux Journalin,
kävelet kirjastoon, valitset lehden,
mutta vasta sitten tarkistat tuliko kirjastokortti mukaan,
jos on,
lainaat lehden,
ellei,
palaat kotiin tyhjin käsin.

While-toistorakenne
Toistorakenteen avulla voidaan suorittaa samoja lauseita toistuvasti peräkkäin.
Toistojen määrä on joko
etukäteen määrätty
tai
se riippuu ohjelmassa esitetyistä ehdoista.



For-toistorakenne
For-rakennetta käytetään, jos toistojen määrä tiedetään ohjelmassa ennen kuin toistot aloitetaan. Toistojen lukumäärää hallitaan toistolaskurin avulla.

Laskuriin kohdistuu kolme operaatiota:
1. laskurin alkuarvon asettaminen
2. laskurin arvon vertaaminen loppuarvoon, ehto
3. laskurin arvon päivittäminen, lisääminen tai vähentäminen yhdellä



For-toistorakenne.

Do ... while –toistorakenne

Jo käsitellyt toistolauseet while ja for ovat molemmat ns. alkuehtoisia toistolauseita. Lauseessa olevan ehdon arvo tutkitaan, ennen kuin siirrytään toistettavaan lauseosaan.

Seuraavaksi käsiteltävä toistolause on loppuehtoinen do ... while -toistorakenne.
Siinä suoritetaan ensin yksi toistokerta ja sen jälkeen tutkitaan pitääkö toistaa uudestaan.