OHJELMOINTIA MATEMATIIKAN OPETUKSEEN

Antti Laaksonen

Kuvitus ja ulkoasu: © Emilia Erfving

Helsinki 2020
ISBN 978-951-51-5938-0 (nid.)
ISBN 978-951-51-5939-7 (PDF)

LUMATIKKA
OPETUSHALLITUS RAHOITTAA HANKETTA
HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI
LUMA-KESKUS SUOMI
LUMA-CENTER FINLAND
LUMA CENTRE FINLAND

Esipuhe

Tämä kirjanen sisältää kokoelman tehtäviä, jotka yhdistävät ohjelmointia ja matematiikkaa eri tavoin. Kokoelman tarkoituksena on antaa ideoita peruskoulun ja lukion matematiikan tunneille ohjelmoinnin sovelluksista.

Tehtävissä jatkuvasti esiintyvä teema on tietokoneen tehokkuus: ohjelmoinnin avulla voimme toistaa asioita automaattisesti hyvin nopeasti. Tämän avulla voimme tutkia monenlaisia matemaattisia ilmiöitä, joita olisi vaikeaa tai mahdotonta lähestyä perinteisin keinoin.

Tutustumme tehtävissä myös tekoälyn toteuttamiseen. Tekoälyt ovat saaneet paljon huomiota viime aikoina, ja ne voivat kuulostaa pelottaviltakin. Todellisuudessa kuitenkin jokaisen tekoälyn takana on ohjelma, joka muodostuu samanlaisista aineksista kuin muutkin ohjelmat.

Jos et ole ohjelmoinut ennen Pythonilla, voit tutustua ensin perusteisiin esimerkiksi sivuston Tie koodariksi (https://tie.koodariksi.fi/) kautta. Sivustolta löydät myös ohjeen, kuinka voit asentaa Pythonin omalle tietokoneellesi.

Ohjelmoinnin oppiminen on seikkailu, jossa taitojen karttuessa tulee jatkuvasti uusia mahdollisuuksia. Tämän kirjasen tehtävät antavat vähän esimakua siitä, mitä ohjelmoinnin avulla pystyy saamaan aikaan!

Helsingissä 20.1.2020
Antti Laaksonen

Kohti algoritmeja

Algoritmi on toimintaohje, jota seuraamalla pääsemme haluttuun lopputulokseen. Usein algoritmin suorittaja on tietokone, mutta algoritmeja voi miettiä mainiosti myös kynällä ja paperilla.

Tehtävä: Luvun muodostaminen

Koneessa on näyttö, jonka ruudulla on aluksi luku 1, sekä kaksi nappia. Vasen nappi lisää lukuun 3 ja oikea nappi kertoo luvun 2:lla.

Esimerkiksi voit muodostaa luvun 17 painamalla kahdesti vasenta nappia, sitten kerran oikeaa nappia ja lopuksi kerran vasenta nappia:

1 → 4 → 7 → 14 → 17

Miten voit muodostaa luvun 16? Entä miten voit muodostaa luvun 100? Mikä olisi yleinen algoritmi, jolla voit muodostaa minkä tahansa annetun positiivisen kokonaisluvun?

Opettajan ohje

Tehtävä: Luvun muodostaminen

Luvun 16 voi muodostaa näin: 1 → 4 → 8 → 16

Luvun 100 voi muodostaa näin: 1 → 4 → 8 → 11 → 22 → 25 → 50 → 100

Saamme aikaan yleisen algoritmin ajattelemalla asiaa käänteisesti. Merkitään x:llä lukua, joka meidän tulee muodostaa. Jos haluamme muodostaa luvun x, sitä ennen meidän täytyy muodostaa luku x – 3 tai x / 2. Jos x on pariton, ainoa mahdollisuus on poistaa 3. Jos taas luku x parillinen, voimme jakaa sen 2:lla. Voimme jatkaa näin eteenpäin, kunnes lopulta saavumme lukuun 1. Esimerkiksi kun haluamme muodostaa luvun 100, edellinen luku on 50, sitä edellinen luku on 25, sitä edellinen luku on 22, sitä edellinen luku on 11, jne.

Tämä toimii aina paitsi silloin, kun haluttu luku on jaollinen 3:lla. Vika ei ole kuitenkaan algoritmissa, vaan tehtävä on mahdoton tässä tapauksessa.

Voimme myös toteuttaa algoritmin näin Pythonilla:

```
print("Anna luku: ")
x = int(input())
luvut = [x]
while x > 1:
    if x%2 == 0:
        x = x//2
    else:
        x = x-3
    luvut = [x]+luvut
print(luvut)
```

Tämä ohjelma kysyy ensin käyttäjältä luvun ja näyttää sitten, kuinka siihen päästään luvusta 1. Ohjelma muodostaa listan, jossa on alussa vain annettu luku x. Sitten ohjelma pienentää lukua silmukassa ja lisää uusia lukuja listan alkuun, kunnes tuloksena on luku 1.

Esimerkiksi ohjelma voi toimia näin:

```
Anna luku: 100
[1, 2, 4, 8, 11, 22, 25, 50, 100]
```

Huomaa, että jos annettu luku x on 3:lla jaollinen, ohjelma toimii väärin, koska ratkaisua ei ole olemassa.

Tietokoneen nopeus

Tietokoneen vahvuutena on, että se laskee *nopeasti*. Esimerkiksi ohjelmoinnissa voimme tehdä silmukan, joka käy läpi nopeasti suuren määrän lukuja.

Esimerkiksi seuraava ohjelma laskee summan 1+2+...+100:

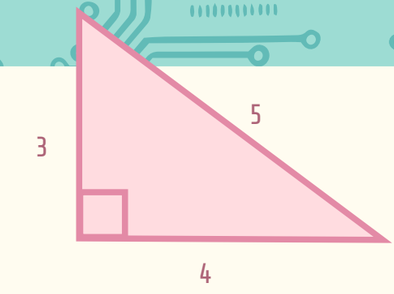
```
summa = 0
for i in range(1,101):
    summa += i
print(summa)
```

Kun ohjelma käynnistetään, se ilmoittaa heti vastauksen 5050.

Tehtävä 1.
Tutkitaan summia

Tietokone pystyy laskemaan nopeasti summan sataan asti (1+2+...+100), mutta mitä käy, jos yläraja on suurempi?

Esimerkiksi kauanko menee laskea summa tuhanteen asti? Entä miljoonaan asti? Missä tietokoneen rajat tulevat vastaan?



Tehtävä 2.
Etsitään kolmioita

Yllä on suorakulmainen kolmio, jonka sivujen pituudet ovat 3, 4 ja 5.

Suorakulmaisessa kolmiossa sivujen pituuksille pätee Pythagoraan lause $a^2 + b^2 = c^2$. Esimerkiksi tässä kolmiossa pätee $3^2 + 4^2 = 5^2$. Tässä kolmiossa myös jokaisen sivun pituus on kokonaisluku.

Löydätkö muita suorakulmaisia kolmioita, joissa jokaisen sivun pituus on kokonaisluku?

Voit etsiä kolmioita kynällä ja paperilla tai ohjelmoimalla. Montakohan tällaista kolmiota on olemassa, joissa jokaisen sivun pituus on enintään 100?

Opettajan ohje

Tehtävä 1.
Tutkitaan summia

Voimme tutkia Python-koodin nopeutta seuraavasti:

```
from time import time

n = 10**6
alku = time()
summa = 0
for i in range(1,n+1):
    summa += i
loppu = time()
print("summa:",summa)
print("aikaa kului:",loppu-alku,"s")
```

Funktio `time` antaa sekuntien määrän tietystä ajanhetkestä alkaen (yleensä vuoden 1970 alusta). Niinpä tallentamalla funktion antama tulos muistiin ennen silmukkaa ja silmukan jälkeen pystymme laskemaan, kauanko suoritus vei aikaa.

Yllä olevassa koodissa muuttujassa *n* on yläraja, johon asti laskemme summaa. Nykypäivän tietokoneella koodi on vielä nopea, kun yläraja on miljoona, mutta vie aikaa, kun yläraja on miljardi.

Tehtävä 2.
Etsitään kolmioita

Pythagoraan kolmikko muodostuu kolmesta positiivisesta kokonaisluvusta (*a*,*b*,*c*), joille pätee $a^2 + b^2 = c^2$. Tehtävässä on siis tavoitteena etsiä Pythagoraan kolmikoita.

Yksi helppo tapa löytää uusia kolmikoita on kertoa olemassa olevan kolmikon jokainen luku jollakin positiivisella kokonaisluvulla *k*: jos (*a*,*b*,*c*) on Pythagoraan kolmikko, niin myös (*ka*,*kb*,*kc*) on Pythagoraan kolmikko. Esimerkiksi kolmikosta (3,4,5) saadaan kolmikko (6,8,10) kertomalla luvulla 2.

Kiinnostavampaa on kuitenkin etsiä aidosti erilaisia uusia kolmikoita. Esimerkkejä tällaisista kolmikoista ovat (5,12,13) ja (8,15,17).

Voimme lähestyä tehtävää ohjelmoinnin kautta käymällä läpi kaikki mahdolliset ratkaisut. Seuraava Python-koodi toteuttaa haun:

```
n = 100
for a in range(1,n+1):
    for b in range(a+1,n+1):
        for c in range(b+1,n+1):
            if a**2 + b**2 == c**2:
                print(a,b,c)
```

Tässä muuttuja *n* antaa ylärajan kolmikossa esiintyvälle luvulle. Koodissa on kolme silmukkaa, jotka käyvät läpi luvut *a*, *b* ja *c*. Jos ehto $a^2 + b^2 = c^2$ pätee, koodi tulostaa ratkaisun.

Koodi löytää 52 ratkaisua, joissa jokainen luku on välillä 1…100. Näistä 16 ovat aidosti erilaisia ratkaisuja. Seuraavassa listassa ovat kaikki ratkaisut, ja aidosti erilaiset on lihavoitu.

(3, 4, 5), **(5, 12, 13)**, (6, 8, 10), **(7, 24, 25)**, **(8, 15, 17)**, (9, 12, 15), **(9, 40, 41)**, (10, 24, 26), **(11, 60, 61)**, (12, 16, 20), **(12, 35, 37)**, **(13, 84, 85)**, (14, 48, 50), (15, 20, 25), (15, 36, 39), (16, 30, 34), **(16, 63, 65)**, (18, 24, 30), (18, 80, 82), **(20, 21, 29)**, (20, 48, 52), (21, 28, 35), (21, 72, 75), (24, 32, 40), (24, 45, 51), (24, 70, 74), (25, 60, 65), (27, 36, 45), **(28, 45, 53)**, (28, 96, 100), (30, 40, 50), (30, 72, 78), **(32, 60, 68)**, (33, 44, 55), **(33, 56, 65)**, (35, 84, 91), (36, 48, 60), (36, 77, 85), (39, 52, 65), (**39, 80, 89**), (40, 42, 58), (40, 75, 85), (42, 56, 70), (45, 60, 75), (**48, 55, 73**), (48, 64, 80), (51, 68, 85), (54, 72, 90), (57, 76, 95), (60, 63, 87), (60, 80, 100), (**65, 72, 97**)

Tekoälyn tekeminen

Tekoäly on ohjelma, joka näyttää toimivan älykkäästi. Kuitenkin tekoäly toimii pohjimmiltaan samalla periaatteella kuin muutkin ohjelmat, eli sen sisällä on muuttujia, ehtoja, silmukoita, jne.

Tekoälyn toteuttajan täytyy miettiä, miten ongelmaa voi lähestyä tietokoneen kannalta niin, että tekoäly toimii riittävän *uskottavasti*. Usein taustalla on enemmän tai vähemmän hämäystä: riittää, että tekoäly *vaikuttaa* toimivan ihmisen tavoin, mutta sen todellinen logiikka voi olla hyvin erilainen.

Tehtävä 1.
Keskusteleva ohjelma

Tässä on pohja ohjelmalle, joka keskustelee käyttäjän kanssa:

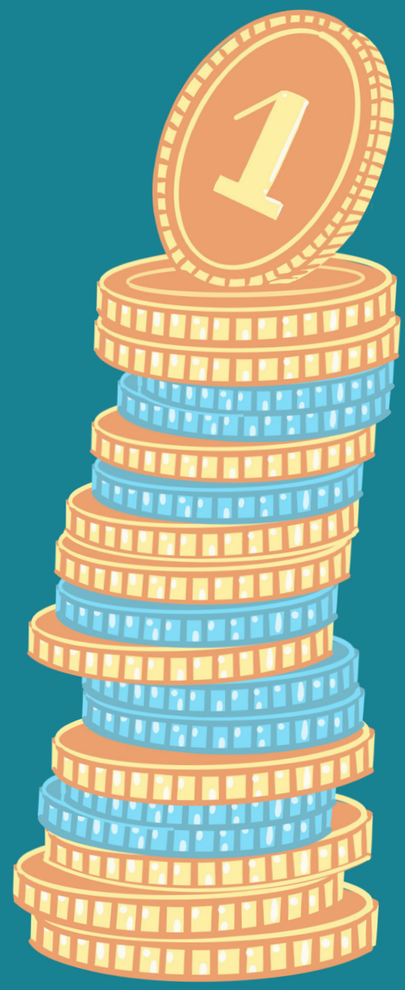
```
from random import *

while True:
    print("Anna kysymys: ")
    kysymys = input()
    sanat = kysymys.split()
    if sanat[0] == "Montako":
        vastaus = randint(1,100)
        print("Luulisin että",vastaus)
    elif sanat[0] == "Oletko":
        vastaus = randint(1,2)
        if vastaus == 1:
            print("Tottakai!")
        else:
            print("En varmasti!")
    else:
        print("Mistä minä sen tietäisin!")
```

Keskustelu ohjelman kanssa voi näyttää tältä:

```
Montako tähteä on taivaalla?
Luulisin että 46
Oletko oikeasti älykäs?
Tottakai!
Mikä on elämän tarkoitus?
Mistä minä sen tietäisin!
```

Tämä ohjelma ei ole vielä kovin hyvä, koska se tuntee niin vähän sanoja ja tilanteita. Koeta laajentaa tekoälyä niin, että se käyttäytyy mahdollisimman aidosti!



Turingin testi

Hyvin toimivan keskustelevan ohjelman tekeminen on keskeinen haaste tekoälyjen tutkimuksessa. Lopullinen koetus tällaiselle ohjelmalle on Turingin testi. Siinä ulkopuolinen koehenkilö keskustelee ohjelman kanssa.

Jos koehenkilö ei pysty sanomaan, onko vastapuolena tekoäly vai toisella koneella oleva ihminen, tekoäly on läpäissyt testin. Tähän mennessä yksikään ohjelma ei ole läpäissyt Turingin testiä.

Tehtävä 2.
Pelin tekoäly

Pelissä on kaksi pelaajaa, jotka tekevät siirtonsa vuorotellen. Pinossa on kolikoita, ja pelaaja saa poistaa vuorollaan 1, 2 tai 3 kolikkoa. Pelin voittaa pelaaja, jonka siirron jälkeen pino on tyhjä.

Tässä on runko ohjelmalle, jossa käyttäjä voi pelata peliä tekoälyä vastaan:

```
from random import *

def tekoaly(k):
    return 1

print("Tervetuloa peliin!")
k = randint(20,30)
while True:
    print("Pinossa on",k,"kolikkoa")
    print("Montako kolikkoa poistat?")
    u = int(input())
    if u not in [1,2,3]:
        print("Ei saa huijata!")
        continue
    k -= u
    if k == 0:
        print("Voitit pelin!")
        break
    u = int(tekoaly(k))
    print("Tekoäly poistaa",u,"kolikkoa")
    k -= u
    if k == 0:
        print("Tekoäly voitti!")
        break
```

Tämä tekoäly on kuitenkin huono, koska se poistaa aina yhden kolikon. Tehtäväsi onkin muuttaa tekoälyä niin, että se toimii paremmin!

Pystyt muuttamaan tekoälyn toimintaa funktiosta `tekoaly`. Sille annetaan parametri `k` (montako kolikkoa on jäljellä) ja sen tulee ilmoittaa, montako kolikkoa tekoäly poistaa siinä tilanteessa.

Opettajan ohje

Tehtävä 1.
Keskusteleva ohjelma

Seuraavat funktiot ovat hyödyllisiä keskustelevan ohjelman tekemisessä:

- Funktio `input` pyytää käyttäjää kirjoittamaan rivin tekstiä. Funktio antaa tekstin merkkijonona, jonka voi sijoittaa muuttujaan näin:

  ```
  teksti = input()
  ```

  Keskustelevan ohjelman pohjassa pyydämme tämän funktion avulla käyttäjältä merkkijonon, joka sisältää tekoälylle annettavan kysymyksen.

- Funktio `split` jakaa merkkijonon osiin välilyöntien kohdilta ja muodostaa listan, joka sisältää merkkijonon jokaisen sanan. Esimerkiksi merkkijonosta `"apina banaani cembalo"` muodostuu lista `["apina","banaani","cembalo"]`. Funktiota voi käyttää vaikkapa näin:

  ```
  sanat = teksti.split()
  ```

  Keskustelevan ohjelman pohjassa jaamme merkkijonon sanoiksi, jotta saamme selville ensimmäisen sanan. Kehittyneempi tekoäly voisi tutkia myös muita kuin ensimmäistä sanaa.

- Funktio `randint(a,b)` arpoo satunnaisen kokonaisluvun väliltä *a*...*b*. Funktion käyttäminen vaatii, että ohjelman alussa on rivi `from random import *` tai vastaava. Esimerkiksi seuraava rivi arpoo muuttujaan vastaus kokonaisluvun väliltä 1...100:

  ```
  vastaus = randint(1,100)
  ```

  Keskustelevan ohjelman pohjassa tuotamme tämän funktion avulla satunnaisen vastauksen kysymykseen, joka alkaa “Montako”.

Tehtävä 2.
Pelin tekoäly

Voimme löytää optimaalisen pelitavan tutkimalla pelin tiloja. Pelin tila tarkoittaa, montako kolikkoa on jäljellä. Osa tiloista on voittotiloja (aloittava pelaaja voittaa), kun taas osa on häviötiloja (toinen pelaaja voittaa).

Tilat 1, 2 ja 3 ovat voittotiloja, koska niissä pelaaja voi poistaa vastaavasti 1, 2 tai 3 kolikkoa ja voittaa pelin. Tila 4 puolestaan on häviötila, koska mikä tahansa siirto johtaa tilaan, joka on vastustajalle voittotila.

Kun tutkimme eteenpäin tiloja, huomaamme, että tässä pelissä tila on häviötila, jos kolikkojen määrä on jaollinen neljällä, ja muuten se on voittotila. Niinpä voimme toteuttaa optimaalisen tekoälyn seuraavasti:

```
def tekoaly(k):
    if k%4 == 0:
        return 1
    else:
        return k%4
```

Ideana on käyttää operaattoria %, joka laskee jakojäännöksen. Jos jakojäännös 4:llä on 0, tila on häviötila ja tekoäly poistaa yhden kolikon toivoen, että vastustaja tekisi virheen. Muuten tila on voittotila ja tekoäly voittaa varmasti poistamalla kolikoita niin, että vastustajalle tulee häviötila.

Tästä pelistä voi keksiä monia muunnelmia muuttamalla, montako kolikkoa saa poistaa. Esimerkiksi yksi kiinnostava muunnelma on peli, jossa joka siirrolla saa poistaa 1, 3 tai 4 kolikkoa (mutta ei 2 kolikkoa).

Todennäköisyys

Todennäköisyyksien laskeminen on usein ihmisille vaikeaa, koska monet tulokset ovat arkijärjen vastaisia. Kuitenkin tietokone pystyy arvioimaan todennäköisyyttä lahjomattomasti simulaation avulla.

Ideana on toistaa ilmiötä suuri määrä kertoja ja tutkia, miten usein haluttu asia tapahtuu. Tämän jälkeen arvion todennäköisyydestä saa kaavalla *k/n*, missä *n* on toistokertojen määrä ja *k* on suotuisten tapahtumien määrä.

Tehtävä 1.
Kolikon heittäminen

Kun heitämme kolikkoa miljoona kertaa, montako kertaa tulee kruuna ja montako kertaa tulee klaava?

Tämä olisi hyvin työlästä tutkia heittämällä kolikkoa itse, mutta onneksi voimme toteuttaa simulaation tietokoneella.

Tee ohjelma, joka arpoo miljoona kertaa kolikon heiton tuloksen ja ilmoittaa lopuksi, montako kertaa tuli kruuna ja klaava!

Tehtävä 2.
Syntymäpäivät

Luokassa on 30 oppilasta, jotka ovat syntyneet samana vuonna. Kuinka todennäköistä on, että kahdella heistä on sama syntymäpäivä?

Mieti ensin itse, miten suuri todennäköisyys voisi olla. Onko se lähempänä 0 vai 100 prosenttia?

Toteuta sitten *simulaatio*, joka antaa arvion todennäköisyydestä. Entä voisiko tuloksen laskea tarkasti matematiikan avulla?

Opettajan ohje

Tehtävä 1.
Kolikon heittäminen

Seuraava koodi toteuttaa simulaation:

```
from random import *

n = 1000000
a = b = 0
for i in range(n):
    t = randint(1,2)
    if t == 1:
        a += 1
    else:
        b += 1
print(a,b)
```

Tässä *n* on heittojen määrä, a on kruunien määrä ja b on klaavojen määrä. Joka heitolla ohjelma arpoo muuttujaan t luvun 1 tai 2. Tulkintana on, että heitto on kruuna, jos luku on 1, ja heitto klaava, jos luku on 2.

Koodi voi antaa vaikkapa seuraavan tuloksen:

```
499455 500545
```

Tällä kertaa klaavoja tuli hieman enemmän, mikä on täysin uskottavaa.

Tehtävä 2.
Syntymäpäivät

Tässä on simulaatio, joka arvio todennäköisyyttä, kun luokassa on tietty määrä oppilaita:

```
from random import *

oppilaat = 30
kierrokset = 1000000
laskuri = 0

for i in range(kierrokset):
    lista = []
    for j in range(oppilaat):
        paiva = randint(1,365)
        if paiva in lista:
            laskuri += 1
            break
        lista.append(paiva)

print(laskuri/kierrokset)
```

Simulaatio muodostuu kierroksista, joista jokainen arpoo oppilaiden syntymäpäivät. Ohjelma olettaa, että vuodessa on 365 päivää. Päivät tallennetaan listaan, ja jos vastaan tulee uudestaan sama päivä, kahdella oppilaalla on sama syntymäpäivä.

Kun oppilaita on 30, miljoonan kierroksen simulaatio antaa tuloksen 0.706673 eli kahdella oppilaalla näyttää olevan sama syntymäpäivä noin 71 prosentin todennäköisyydellä. Tämä on yllättävän suurelta tuntuva todennäköisyys, ja ilmiö tunnetaan nimellä *syntymäpäiväparadoksi*.

Voimme laskea tuloksen myös matemaattisesti kaavalla

$$1 - 1 \cdot \left(\frac{364}{365}\right) \cdot \left(\frac{363}{365}\right) \cdot \left(\frac{362}{365}\right) \ldots,$$

missä kertolaskun osien määrä on sama kuin oppilaiden määrä. Ideana on, että kertolasku antaa todennäköisyyden sille, että kaikilla oppilailla on eri syntymäpäivä. Sitten kun tämä vähennetään luvusta 1, saadaan haluttu todennäköisyys.

Esimerkiksi kun oppilaita on 30, kaava antaa tuloksen 0.706316. Saimme siis varsin tarkan tuloksen myös simulaation avulla.

Kielen tunnistaminen

Kuinka voisi tunnistaa *automaattisesti*, millä kielellä teksti on kirjoitettu?

Yksi lähestymistapa perustuu havaintoon, että eri kielillä kirjoitetuissa teksteissä on eri *vokaalisuhde*. Vokaalisuhde tarkoittaa, kuinka suuri osa tekstissä olevista kirjaimista on vokaaleita. Se lasketaan kaavalla $v/(v+k)$, missä v on vokaalien määrä ja k on konsonanttien määrä. Esimerkiksi sanan “apina” vokaalisuhde on 3/5 = 0.60.

Tutkimalla tekstiaineistoja voidaan selvittää, mitä vokaalisuhteita eri kielissä esiintyy. Suomen kielessä vokaalisuhde on noin 0.48, kun taas englannin kielessä se on noin 0.40. Tämä tarkoittaa, että suomessa on enemmän vokaaleita kuin englannissa. Voimmekin veikata, että teksti on suomea, jos vokaalisuhde on lähempänä lukua 0.48 kuin lukua 0.40, ja muuten englantia.

Tarkastellaan esimerkkinä seuraavia tekstinpätkiä:

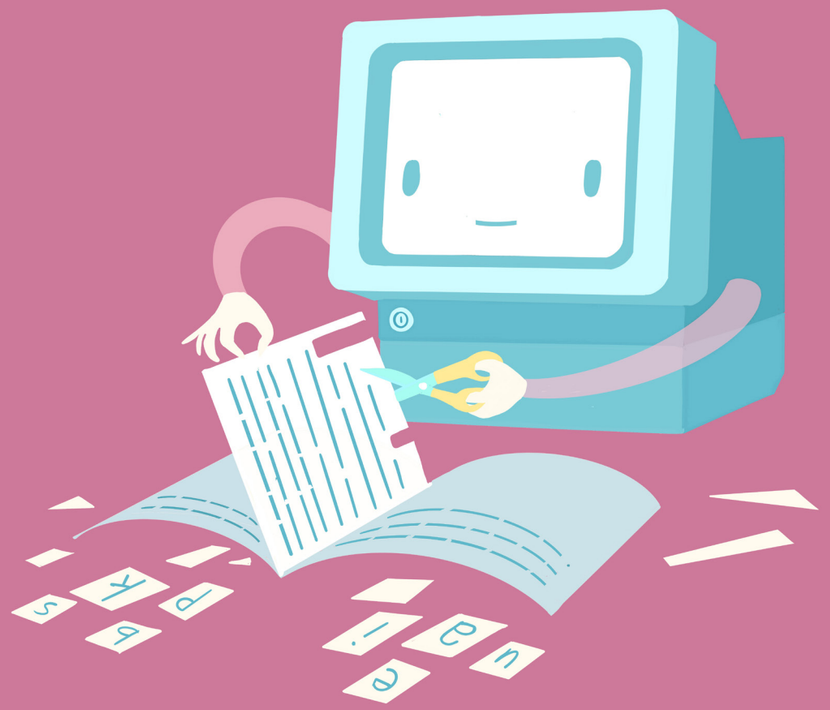
Teksti 1:

”Apinalla on kolme banaania
ja se soittaa cembaloa.”

Teksti 2:

“A monkey has three bananas and
it plays the harpsichord.”

Tekstissä 1 on 22 vokaalia ja 20 konsonanttia, joten vokaalisuhde on 0.52. Tekstissä 2 on puolestaan 18 vokaalia ja 28 konsonanttia, joten vokaalisuhde on 0.39. Tämän perusteella voimme päätellä, että teksti 1 on suomea, kun taas teksti 2 on englantia.



Tehtävä:
Suomea vai englantia?

Tutki, miten luotettavasti vokaalisuhteesta pystyy päättelemään, onko teksti suomea vai englantia!

Miten tunnistamista voisi huijata? Keksitkö suomenkielisen tekstin, jonka vokaalisuhde viittaa englantiin, ja englanninkielisen tekstin, jonka vokaalisuhde viittaa suomeen?

Project Gutenbergissä (https://www.gutenberg.org/) on erikielisiä kirjoja. Millaisia vokaalisuhteita niissä on? Entä muissa kielissä kuin suomessa ja englannissa?

Opettajan ohje

Tehtävä: Suomea vai englantia?

Seuraava koodi laskee tiedostossa `teksti.txt` olevan tekstin vokaalisuhteen:

```
v = k = 0
rivit = open("teksti.txt").readlines()
for r in rivit:
    r = r.lower()
    for m in r:
        if m in "aeiouyääö":
            v += 1
        if m in "bcdfghjklmnpqrstvwxz":
            k += 1
print("Vokaalit:",v)
print("Konsonantit:",k)
print("Vokaalisuhde:",v/(v+k))
```

Koodi luo ensin listan `rivit`, johon luetaan tiedoston rivit. Tämän jälkeen koodi käy läpi jokaisen rivin ja muuttaa sen kaikki kirjaimet pieniksi `lower`-funktiolla. Sitten koodi käy rivin merkit läpi ja pitää kirjaa vokaalien ja konsonanttien määristä.

Esimerkiksi kun tekstinä on “Apinalla on kolme banaania ja se soittaa cembaloa”, ohjelman tulostus on seuraava:

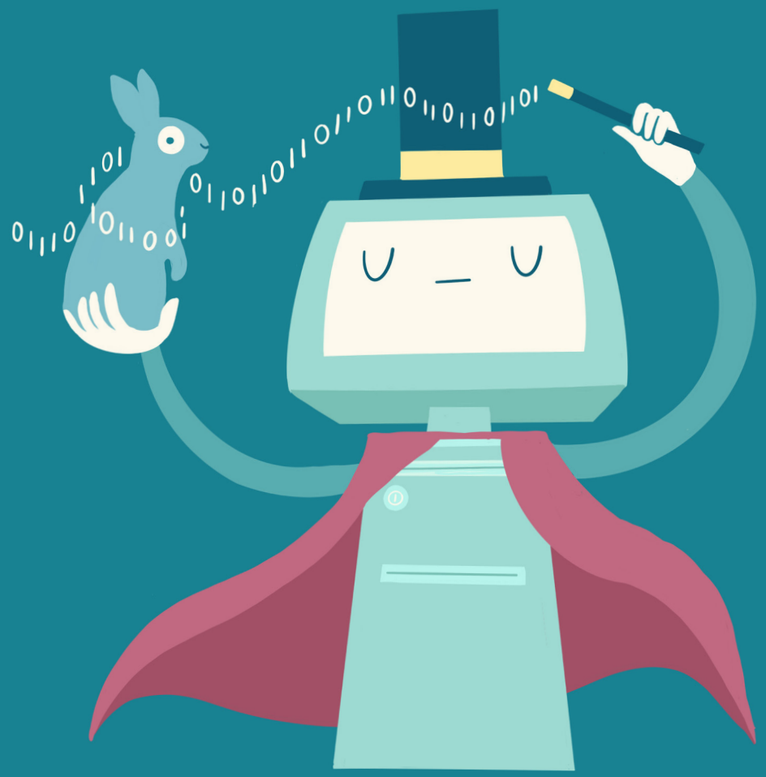
```
Vokaalit: 22
Konsonantit: 20
Vokaalisuhde: 0.5238095238095238
```

```
https://www.gutenberg.org/
```

Project Gutenbergistä löytyy esimerkiksi Kalevala sekä sen käännöksiä. Kirjat pystyy lataamaan tekstitiedostoina (*Plain Text UTF-8*), jotka pystyy antamaan suoraan ohjelmalle. Voimme laskea tiedostoista esimerkiksi seuraavat vokaalisuhteet:

- Suomenkielinen kirja: 0.47464
- Englanninkielinen kirja: 0.38309

Huomaa, että tiedostojen alussa ja lopussa on kirjaan kuulumatonta tekstiä. Tämän osuus kokonaisuudesta on kuitenkin pieni, eikä se vaikuta paljon vokaalisuhteeseen. Jos haluaisimme laskea tarkemmin, voisimme kuitenkin poistaa nämä osuudet.



Bittien taikaa

Tietokoneen sisällä jokainen luku on tallennettu bitteinä. Esimerkiksi bittiesitys 11010 tarkoittaa lukua 26, koska ykkösbitit vastaavat lukuja 16, 8 ja 2 ja laskemalla ne yhteen saadaan 16 + 8 + 2 = 26. Kun käytössä on *n* bittiä, niillä voi esittää 2^n erilaista lukua, koska jokainen bitti voi olla 0 tai 1.

Yksi tapa ajatella bittejä on, että jokainen bitti on vastaus kyllä/ei-kysymykseen. Tähän perustuu esimerkiksi seuraava taikatemppu.

Tehtävä: Iän arvaaminen

Sudenpentujen käsikirjassa on annettu menetelmä, jonka avulla voi selvittää henkilön iän ovelalla tavalla. Se perustuu seuraaviin tauluihin:

Taulu 1
1 3 5 7 9 11 13 15 17
19 21 23 25 27 29 31
33 35 37 39 41 43 45
47 49 51 53 55 57 59
61 63 65 67 69 71 73
75 77 79 81 83 85 87
89 91 93 95 97 99
101 103 105 107

Taulu 2
2 3 6 7 10 11 14 15 18
19 22 23 26 27 30 31
34 35 38 39 42 43 46
47 50 51 54 55 58 59
62 63 66 67 70 71 74
75 78 79 82 83 86 87
90 91 94 95 98 99 102
103 106 107

Taulu 3
4 5 6 7 12 13 14 15 20
21 22 23 28 29 30 31
36 37 38 39 44 45 46
47 52 53 54 55 60 61
62 63 68 69 70 71 76
77 78 79 84 85 86 87
92 93 94 95 100 101
102 103

Taulu 4
8 9 10 11 12 13 14 15
24 25 26 27 28 29 30
31 40 41 42 43 44 45
46 47 56 57 58 59 60
61 62 63 72 73 74 75
76 77 78 79 88 89 90
91 92 93 94 95 104
105 106 107

Taulu 5
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30 31 48 49 50 51 52
53 54 55 56 57 58 59
60 61 62 63 80 81 82
83 84 85 86 87 88 89
90 91 92 93 94 95

Taulu 6
32 33 34 35 36 37 38
39 40 41 42 43 44 45
46 47 48 49 50 51 52
53 54 55 56 57 58 59
60 61 62 63 96 97 98
99 100 101 102 103
104 105 106 107

Taulu 7
64 65 66 67 68 69 70
71 72 73 74 75 76 77
78 79 80 81 82 83 84
85 86 87 88 89 90 91
92 93 94 95 96 97 98
99 100 101 102 103
104 105 106 107

Kun haluat tietää henkilön iän, pyydä häntä ilmoittamaan, missä tauluissa ikä esiintyy. Tämän jälkeen saat iän selville laskemalla yhteen ensimmäiset luvut valituista tauluista.

Esimerkki: Henkilö ilmoittaa, että hänen ikänsä esiintyy tauluissa 1, 2 ja 6. Näissä tauluissa ensimmäiset luvut ovat 1, 2 ja 32, joten henkilön ikä on 1 + 2 + 32 = 35 vuotta.

Kokeile, toimiiko menetelmä omalla iälläsi ja jonkun toisen iällä!

Mihin menetelmä perustuu? Miksi iän saa selville laskemalla yhteen ensimmäiset luvut tauluista?

Opettajan ohje

Tehtävä:
Iän arvaaminen

Taikatempussa käytetty menetelmä perustuu siihen, että mikä tahansa positiivinen kokonaisluku voidaan esittää yksikäsitteisesti summana kakkosen potensseja. Esimerkiksi $35 = 2^0 + 2^1 + 2^5$. Jokaisessa taulussa pienin luku on kakkosen potenssi, joka esiintyy kaikissa taulussa olevissa luvuissa.

Menetelmän voi ymmärtää käsittelemällä lukuja bittimuodossa. Tällöin taulussa x ovat ne luvut, joissa bitti x oikealta laskien on 1. Esimerkiksi luku 35 on bittimuodossa 100011.

Iän arvaamisen voi toteuttaa näin ohjelmoimalla:

```
ika = 0
for x in range(7):
    print("Taulu",x+1)
    taulu = []
    for i in range(1,108):
        if i&(1<<x) != 0:
            taulu.append(str(i))
    print(" ".join(taulu))
    print("Onko ikäsi tässä taulussa? (K/E)")
    if input() == "K":
        ika += 1<<x
print("Ikäsi on:",ika)
```

Taulu 1
1 3 5 7 9 11 13 15 17
19 21 23 25 27 29 31
33 35 37 39 41 43 45
47 49 51 53 55 57 59
61 63 65 67 69 71 73
75 77 79 81 83 85 87
89 91 93 95 97 99
101 103 105 107

Taulu 2
2 3 6 7 10 11 14 15 18
19 22 23 26 27 30 31
34 35 38 39 42 43 46
47 50 51 54 55 58 59
62 63 66 67 70 71 74
75 78 79 82 83 86 87
90 91 94 95 98 99 102
103 106 107

Taulu 3
4 5 6 7 12 13 14 15 20
21 22 23 28 29 30 31
36 37 38 39 44 45 46
47 52 53 54 55 60 61
62 63 68 69 70 71 76
77 78 79 84 85 86 87
92 93 94 95 100 101
102 103

Taulu 4
8 9 10 11 12 13 14 15
24 25 26 27 28 29 30
31 40 41 42 43 44 45
46 47 56 57 58 59 60
61 62 63 72 73 74 75
76 77 78 79 88 89 90
91 92 93 94 95 104
105 106 107

Taulu 5
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30 31 48 49 50 51 52
53 54 55 56 57 58 59
60 61 62 63 80 81 82
83 84 85 86 87 88 89
90 91 92 93 94 95

Taulu 6
32 33 34 35 36 37 38
39 40 41 42 43 44 45
46 47 48 49 50 51 52
53 54 55 56 57 58 59
60 61 62 63 96 97 98
99 100 101 102 103
104 105 106 107

Taulu 7
64 65 66 67 68 69 70
71 72 73 74 75 76 77
78 79 80 81 82 83 84
85 86 87 88 89 90 91
92 93 94 95 96 97 98
99 100 101 102 103
104 105 106 107

Ohjelma käy läpi taulut ja kysyy jokaisen taulun kohdalla käyttäjältä, onko hänen ikänsä taulussa. Merkintä 1<<x tarkoittaa bitin 1 siirtämistä x askelta vasemmalle eli lukua 2^x. Merkintä & tarkoittaa and-operaatiota, jonka avulla voi tarkastaa, onko tietty luvun bitti 1.

Tämän tehtävän opetuksena on, miten paljon tietoa muutaman kyllä/ei-kysymyksen avulla voi selvittää. Jokainen kysymys kaksinkertaistaa tiedon määrän, eli esimerkiksi 7 kysymyksellä voi selvittää vastauksen $2^7 = 128$ vaihtoehdon joukosta. Tämä riittää henkilön iän selvittämiseen (tässä iän ylärajana on 107, mutta voisi olla jopa 127 ilman, että tauluja tarvittaisiin enemmän).

Ohjelmoinnin mahdollisuudet

Syvällisen ohjelmointitaidon hankkiminen vie paljon aikaa, mutta siitä saatava palkintokin on suuri: sen jälkeen tietokoneella voi tutkia mitä tahansa ilmiötä, jonka pystyy esittämään matemaattisesti.

Seuraavassa on yksi selvästi vaikeampi tehtävä, joka sopii haasteeksi kattavasti ohjelmointiin perehtyneelle.

Tehtävä: Latinalaiset neliöt

Latinalainen neliö on n x n -ruudukko, jonka jokaisessa ruudussa on luku väliltä 1...n. Lisäksi jokainen luku esiintyy tasan kerran kullakin pysty- ja vaakarivillä. Latinalainen neliö muistuttaa siis tuttua sudoku-tehtävää, mutta siinä n voi olla mikä tahansa eikä ole pikkuneliöitä.

Esimerkiksi kun $n = 3$, voimme muodostaa latinalaisen neliön 12 tavalla:

1	2	3
2	3	1
3	1	2

1	2	3
3	1	2
2	3	1

1	3	2
2	1	3
3	2	1

1	3	2
3	2	1
2	1	3

2	1	3
1	3	2
3	2	1

2	1	3
3	2	1
1	3	2

2	3	1
1	2	3
3	1	2

2	3	1
3	1	2
1	2	3

3	1	2
1	2	3
2	3	1

3	1	2
2	3	1
1	2	3

3	2	1
1	3	2
2	1	3

3	2	1
2	1	3
1	3	2

Entä kun n on suurempi? Monellako tavalla latinalaisen neliön voi muodostaa, kun $n = 4$, $n = 5$, tai vielä suuremmissa tapauksissa?

Osaatko laatia ohjelman, joka laskee vastauksen annetulle n:n arvolle?

Opettajan ohje

Tehtävä: Latinalaiset neliöt

Seuraava ohjelma laskee, montako *n* x *n* -kokoista latinalaista neliötä on olemassa:

```
n = 4
vastaus = 0
vaaka = {}
pysty = {}

def laske(y,x):
    global vastaus
    if y == n+1:
        vastaus += 1
    elif x == n+1:
        laske(y+1,1)
    else:
        for i in range(1,n+1):
            if (y,i) in vaaka or (x,i) in pysty:
                continue
            vaaka[(y,i)] = pysty[(x,i)] = True
            laske(y,x+1)
            del vaaka[(y,i)], pysty[(x,i)]

laske(1,1)
print(vastaus)
```

Ohjelman alussa annetaan ruudukon koko *n*. Funktio `laske` selvittää vastauksen rekursiivisesti käymällä läpi kaikki mahdolliset ruudukot. Joka kutsulla funktio valitsee ruudukon kohtaan (*y*,*x*) tulevan luvun. Jotta ruudukosta tulee latinalainen neliö, rakenteet `vaaka` ja `pysty` pitävät muistissa, mitä lukuja milläkin vaaka- ja pystyrivillä on jo valmiina.

Suorittamalla ohjelma selviää, että 4 x 4 -kokoisia latinalaisia neliöitä on 576 ja 5 x 5 -kokoisia neliöitä on peräti 161280. Neliöiden määrä kasvaa räjähdysmäisesti, kun *n*:n arvo suurenee.

Rekursio on yleinen menetelmä, jonka avulla voimme ratkaista järjestelmällisesti minkä tahansa vastaavan ongelman, koska voimme käydä läpi kaikki vaihtoehdot ratkaisun muodostamiseen.

Pieni ohjelmointisanasto

Bugi

Ohjelman koodissa oleva virhe. Bugin takia ohjelma voi antaa outoja tuloksia tai ei ehkä käynnisty ollenkaan.

Ehto

Ehdon avulla ohjelman saa toimimaan eri tavalla eri tilanteissa. Esimerkiksi ehto x>5 vaatii, että muuttujan x sisältö on yli 5.

Funktio

Koodin osa, jolle on annettu tietty nimi. Funktiolle voidaan antaa tietoa parametreina ja se voi palauttaa lopuksi jonkin tuloksen.

Lista

Listan avulla voimme pitää muistissa paljon tietoa. Esimerkiksi [4,2,5,4,1] on lista, joka sisältää viisi kokonaislukua.

Merkkijono

Merkkijono tarkoittaa samaa kuin teksti. Esimerkiksi ”apina” on merkkijono, joka muodostuu viidestä merkistä.

Muuttuja

Muuttuja on paikka, jossa ohjelma voi säilyttää tietoa. Muuttujalla on ohjelmoijan valitsema nimi, jolla siihen viitataan koodissa.

Silmukka

Silmukan avulla ohjelma voi toistaa samaa koodia monta kertaa. Tavallisia silmukoita ovat `for` ja `while`.

Tulostaminen

Tulostaminen tarkoittaa, että ohjelma laittaa tekstiä näkyviin näytölle. Tavallinen tapa tulostaa on käyttää komentoa `print`.