Perusmatematiikkaa ohjelmoimalla

Yläkoulun matematiikan lisämateriaali

Jouko Järvenpää

Perusmatematiikkaa ohjelmoimalla

Yläkoulun matematiikan lisämateriaali

Suomen tietokirjailijat ry ja Otavan Kirjasäätiö ovat tukeneet hanketta.

Salo 2021

ISBN 978-952-94-4547-9 (PDF)

Lukijalle

Perusmatematiikkaa ohjelmoimalla -teos sisältää peruskoulun vuosiluokille 7. – 9. ohjelmoinnillisia matematiikan tehtäviä. Teos on jaettu kolmeen harjoituskirjaan. Kullekin vuosiluokalle on oma harjoituskirja. Tehtävissä opit Python-ohjelmoinnin perusteet ja samalla vahvistat matemaattista osaamistasi.

Tehtäviä voit hyvin käyttää matematiikan teorian kertaamiseen, lisätehtävinä tai niitä voit tehdä etätehtävinä kotona. Tehtävien tekemiseksi tarvitset ilmaisen **Python**-ohjelmointiympäristön version **3.x** (mieluiten 3.8 tai uudempi). Sen voit ladata osoitteesta https://www.python.org/downloads/.

Pythonin asentaminen on helppoa. Asennuspaketissa on mukana koodaamisessa tarvittava Idle-editori. Idle-editoria on käytetty myös teorian ja tehtävien mallikuvissa.

Tämän materiaalin lisäksi kannattaa selailla netistä löytyvää Python-materiaalia. Esimerkiksi seuraavista tietolähteistä löytyy paljon hyödyllistä perustietoa Python-ohjelmoinnista:

- ✓ https://www.python.org/doc/
- ✓ https://www.w3schools.com/python/

Ilman Python-ohjelmiston asentamista koodaaminen on mahdollista selainohjelmassa mm. seuraavissa osoitteissa:

- ✓ https://www.jdoodle.com/python3-programming-online/
- ✓ https://www.tutorialspoint.com/execute_python_online.php

Muistathan seuraavat neuvot koodaustehtäviä tehdessäsi:

- Lue teoriasivu huolella ja tutustu esimerkkiohjelmiin niin, että ymmärrät niiden toiminnan.
- Aloita helpoista tehtävistä ja etene vaikeampiin.
- Ole sitkeä. Älä luovuta heti. Mieti, mitä osia osaat tehdä tehtävästä. Kirjoita ne ensin ja mieti sitten vaikeammat kohdat.
- Kysy apua ja auta myös kaveria. Näin opit parhaiten.

Hauskoja koodaushetkiä

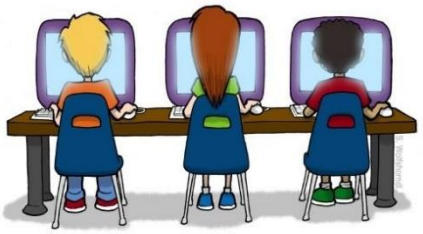
Jouko Järvenpää

PERUSMATEMATIIKKAA OHJELMOIMALLA

matemaattisen ohjelmoinnin harjoituskirja 7-luokalle

Oppilaan nimi:_____

Python-ohjelmointia 7. vuosiluokan matematiikassa



Ohjelmoitavat elektroniset laitteet, arkielämän sähköiset palvelut, mobiiliapplikaatiot, sovellusohjelmat, tietokonepelit, elokuvien tehosteet, musiikkituotanto, taloautomaatio, kaupankäynti, rahaliikenne, pankkipalvelut, teollisuusautomaatio, liikenteenohjaus, turvallisuuden valvonta, sotilasteknologia ja niin edelleen. Tietokonetekniikkaa ja ohjelmoituja sovelluksia on lähes kaikkialla. Laitteiden ja ohjelmien toiminnan ymmärtämiseksi pitää ymmärtää ohjelmoinnin perusasioita.

Tämän tehtäväkirjan avulla opit Python-ohjelmoinnin perusteet. Tehtävissä sovelletaan usein matematiikasta tuttuja asioita ja käsitteitä, joten samalla vahvistat myös matematiikan osaamistasi.

Ohjelmointi kehittää keskittymiskykyä, matemaattisten mallien ajattelutaitoa sekä algoritmista eli vaiheittaista ongelman ratkaisutaitoa. Ohjelmointi voi aluksi tuntua hankalalta, varsinkin, jos et ole kovin kokenut tietokoneen käytössä. Voit kuitenkin uskoa, että muutaman tehtävän tehtyäsi ja tutustuttuasi rohkeasti ohjelmointiin, huomaat, että opit nopeasti ohjelmoinnillisen ajattelutavan. Huomaat myös, miten yksinkertaiset ja yksityiskohtaiset ohjeet tietokone tarvitsee selviytyäkseen tehtävästä.

Tehtäväluettelo:

1. Miten tietokone toimii? Mitä ohjelmointi on?

Opit uutta:

- tietokoneen toiminnan perusperiaate
- syöttö– ja tulostuslaitteet
- ohjelmoinnin periaate, algoritmi
- ohjelman suoritus tietokoneessa

Osaat jo:

- käynnistää tietokoneen ja jonkin ohjelman
- tallentaa tiedoston kansioon
- avata tallennetun tiedoston kansiosta
- käyttää sujuvasti näppäimistöä ja hiirtä

Tietokoneeseen syötetään tietoa ja vaikutetaan ohjelmien toimintaan **syöttölaitteiden** avulla (näppäimistö, hiiri, kosketusnäyttö. Tietokone prosessoi syötettyjä tietoja prosessorin, sisäisen muistin ja ohjelmassa annettujen ohjeiden avulla. Ohjelman ja tietokoneen tuottamat tulokset saadaan **tulostuslaitteiden** avulla (näyttö, kaiuttimet, tulostin).

Tietokone käsittelee tietoja **bitteinä**; ykkösten ja nollien yhdistelminä. Esimerkiksi **binaariluku** 1011 on kymmenjärjestelmän lukuna 11. Käyttäjä voi kuitenkin syöttää koneelle tietoja ihmiselle selkeämmässä esitysmuodossa esimerkiksi aakkosmerkeillä ja kymmenjärjestelmän lukuina sekä hiirellä klikkailemalla.

Esimerkiksi lettujen valmistusohje on arkikielinen esimerkki algoritmista. Kun ohjeen vaiheet suoritetaan annetun mukaisesti, saadaan ohjelman suorituksen tuloksena maistuvia lettuja.

Matemaattiset laskutoimitukset noudattavat tarkoin määrättyä algoritmia. Ohessa on Python-kielellä ohjelmoitu koodi, joka laskee suorakulmion pinta-alan. Se ei eroa juurikaan matemaattisesti laaditusta ratkaisusta.

```
# Tämä on kommenttirivi ja se aloitetaan #-merkillä
# Pythonissa kommenttirivi ei vaikuta ohjelman toimintaan.
# Lasketaan suorakulmion pinta-ala.

a = 4 # Asetetaan muuttujalle a arvoksi 4
b = 3 # Asetetaan muuttujalle b arvoksi 3
Ala = a * b # Lasketaan pinta-ala ja asetetaan muuttujaan Ala

print ("Suorakulmion pinta-ala on",Ala)
```

Tiedon syöttö sekä ja ohjelmien toiminnan ohjaaminen tapahtuu **syöttölaitteiden** avulla (näppäimistö, hiiri, kosketusnäyttö). Tietokone prosessoi syötettyjä tietoja prosessorin, sisäisen työmuistin (RAM-muisti) ja ohjelmassa annettujen ohjeiden avulla. Ohjelman ja tietokoneen tuottamat tulokset saadaan näkyviin **tulostuslaitteiden** (näyttö, kaiuttimet, tulostin) avulla.

Leivonnaisten ja ruokareseptien kirjoittaminen tai leikkipalikoiden rakennusohjeet ovat ohjelmia. Tietokoneen ohjelmoinnissa kirjoitamme tietokoneelle yksikäsitteiset, selkeät ja vaiheittaiset ohjeet jonkin tehtävän suorittamiseen. Tehtävän tai ongelman ratkaisevaa koodia kutsutaan **algoritmiksi**. Algoritmin tulee noudattaa ohjelmointikielen sääntöjä eli **syntaksia**.

Tietokoneohjelmat kirjoitetaan aina jollakin ohjelmointikielellä. Kielen valintamahdollisuuksia on kymmeniä. Tietokoneessa korkean tason ohjelma (esim. Python-koodi) muunnetaan **konekieliseksi mikrokoodiksi** eli bittijonoksi. Ohjelma tallennetaan tietokoneen muistiin. Muistista prosessori suorittaa konekielisen ohjelman bitteinä ja tulokset siirretään väylää pitkin tulostuslaitteelle.

Usein tehtävään valitaan ratkaisemiseen parhaiten soveltuva kieli. Python-kieli soveltuu yksinkertaisen esitystavan ja selkeän rakenteen vuoksi hyvin ohjelmoinnin opiskeluun.

TEHTÄVIÄ

Tehtävä 1. Tiedonhakua ohjelmointikielistä

Etsi internetistä tietoa ja kerro lyhyesti, mitkä ovat suosituimmat ohjelmointikielet.

_____

_____

Mikä on Python-kielen sijoitus suosituimpien ohjelmointikielten listassa?

Tehtävä 2. Tiedonhakua ohjelmoinnista

Etsi internetistä vastaukset seuraaviin kysymyksiin. Vastaa lyhyesti yhdellä lauseella.

Mikä on ohjelmointikielen tulkki?

_____

Onko Python tulkattava vai käännettävä ohjelmointikieli?

_____

Tehtävä 3. Ohjelmoitavissa laitteissa on prosessori

Luettele muutamia prosessoriohjattuja laitteita, joiden käyttö perustuu johonkin ohjelmoituun toimintaan.

_____

_____

_____

Mitä ohjelmoitavia eli prosessoriohjattuja laitteita sinulla on käytössäsi?

_____

Tehtävä 4. Valmistetaan kaakaota

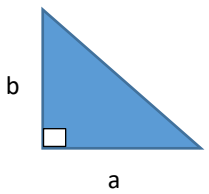
Kirjoita **arkikielinen** ohje (algoritmi) kaakaon valmistamiseksi. Ohjeen tuloksena tulee saada kaakaojauheesta kupillinen maitokaakaota. (Kaikkia rivejä ei välttämättä tarvita)

1. _____
2. _____
3. _____
4. _____
5. _____
6. _____
7. _____

Tehtävä 5. **Algoritmit matematiikassa**

Matematiikan laskulausekkeet ovat myös algoritmeja. Kirjoita matemaattinen algoritmi suorakulmaisen kolmion pinta-alan laskemiseksi.

Ala =



Tehtävä 6. Etunimet aakkosjärjestykseen

Kirjoita mahdollisimman tarkka algoritmi, jolla saadaan järjestettyä aakkosjärjestykseen kolme nimeä *Assi, Aimo, Aki*. (Kaikkia rivejä ei välttämättä tarvita)

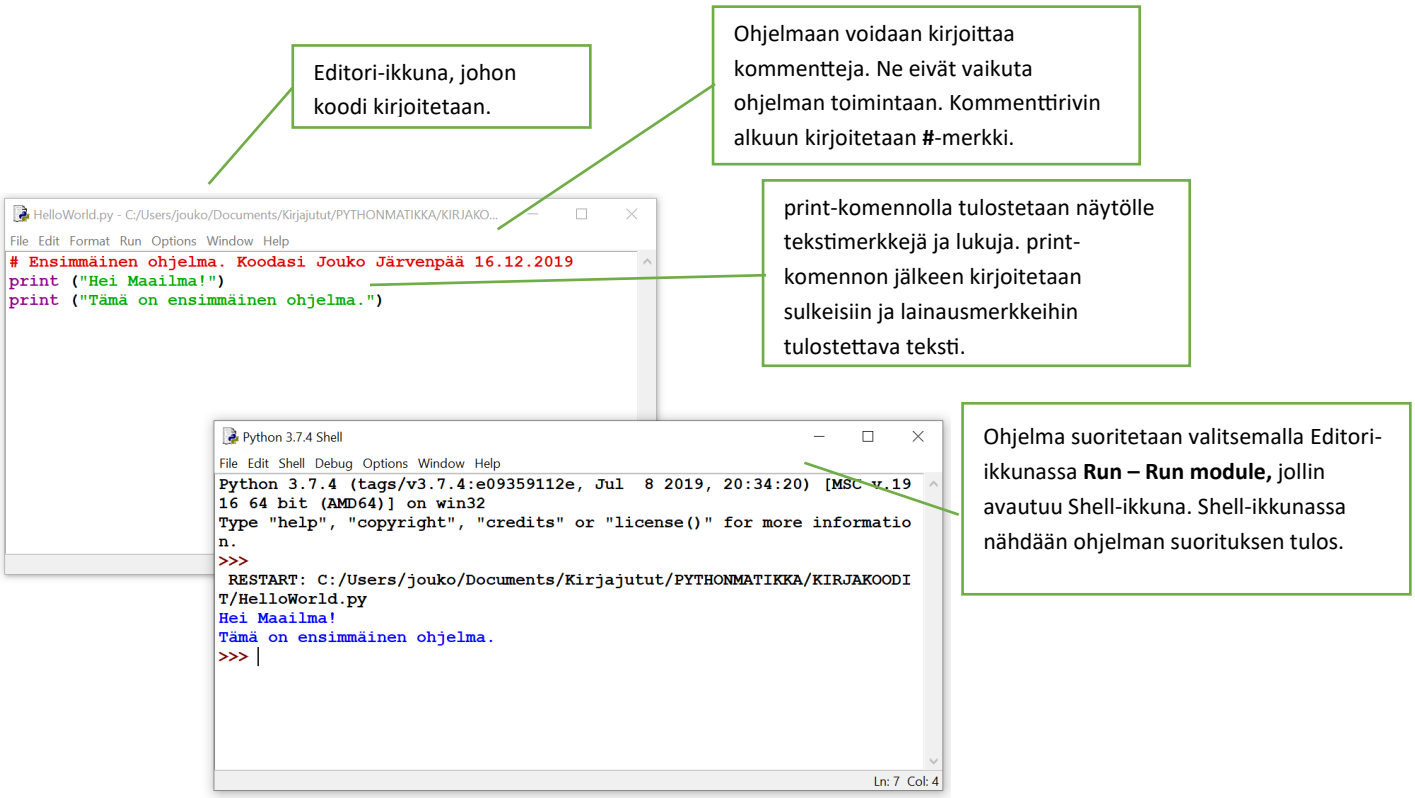
1. _____
2. _____
3. _____
4. _____
5. _____
6. _____
7. _____

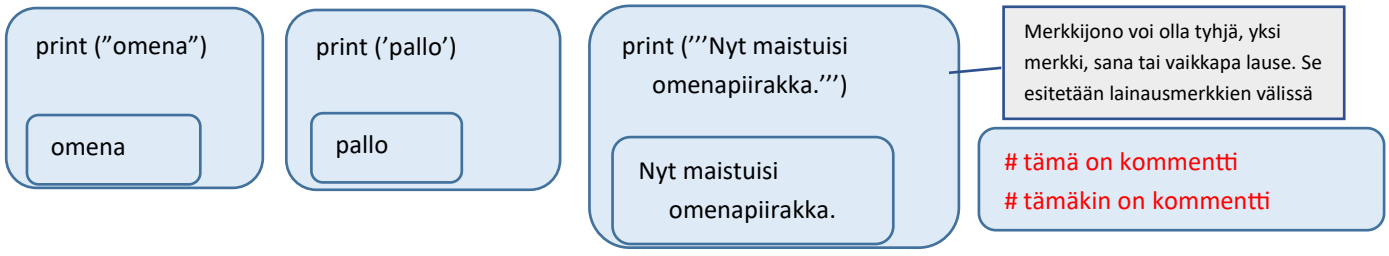
2. Ensimmäinen Python-ohjelma, print-komento

Opit uutta:
- Idle-editorin käynnistäminen ja käyttäminen
- koodin tallentaminen ja tallennetun koodin avaaminen
- print-komento
- kommentit ohjelmassa

Osaat jo:
- kirjoittaa yksinkertaisen algoritmin arkikielellä
- mitä ohjelmointi tarkoittaa
- syöttö- ja tulostuslaitteet

Ohjelmakoodi kirjoitetaan jollakin **tekstieditorilla**. Soveltuvia editoreita ovat niin sanotut *plain text* -editorit (Windowsin Muistio, Notepad++, TextEdit, Idle). Tekstinkäsittelyohjelmat (esim. Word) eivät sovellu koodin kirjoittamiseen, koska tekstissä on mukana Python tulkille epäsopivaa muotoilua. Valmis koodi tallennetaan tiedostoon (**File – Save As...**) ja nimetään. Tiedosto on hyvä nimetä ohjelman toimintaa kuvaavalla tavalla, esimerkiksi *listanlajittelu*. Tallennuksen jälkeen koodi voidaan suorittaa (Run – Run Module). Alla kuvassa Pythonin mukana tulevan **Idle**-editorin ikkunat.





Pythonissa tekstin, lukujen ja laskutoimitusten tulostaminen näytölle tehdään **print**-komennolla. Print-komennossa tulostettava merkkijono kirjoitetaan sulkeisiin lainausmerkkien sisään. Kelvollisia lainausmerkkejä ovat kokolainausmerkki (”omena”) tai puolilainausmerkki (’pallo’). Kirjoittamalla koko- tai puolilainausmerkkejä kolmasti, voidaan tulostettavaa tekstiä rivittää ja sisentää.

Hyvään koodaamiseen kuluu kommentointi. Pythonissa kommenttirivin aloitusmerkki on **#**. Ohjelmakoodin kommentteihin kannattaa kirjoittaa, mitä ohjelmassa tehdään. Kommentointi helpottaa ohjelman päivitystä myöhemmin ja selventää koodin toimintaa ulkopuoliselle.

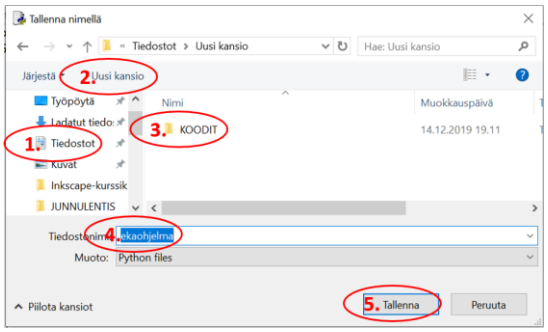
TEHTÄVIÄ

Tehtävä 7. Ensimmäinen Python-ohjelma

1. Käynnistä IDLE-editori ja valitse Shell-ikkunan ylävalikosta **File – New File**.

2. Kirjoita uuteen tiedostoon oheisen mallin mukaiset koodit.
 # ensimmäinen ohjelmani, oma nimesi ja päivämäärä
print ("Hei maailma")
print ("Tämä on ensimmäinen Python-koodini")

3. Tallenna tiedosto KOODIT- kansioon valitsemalla File – Save as... Tallennuskansion luonti ja tiedoston nimeämisvaiheet näkyvät oheisesta kuvasta. Tallenna koodi nimellä *ekaohjelma*.



4. Suorita ohjelma valitsemalla Run – Run module.

Tehtävä 8. print-komennolla tulostetaan näytölle

Aloita uusi tiedosto valitsemalla File – New File

Koodaa ohjelma, joka tulostaa koko nimesi yhdelle riville.

Tallenna koodi KOODIT-kansioon nimellä *nimentulostus*.

Suorita ohjelma valitsemalla Run – Run Module tai painamalla näppäimistön F5-näppäintä.

Minna Annikki Leino

Tehtävä 9. Virheet tutuksi

Kirjoita tarkoituksella syntaksivirhe ohjelmakoodiin ja tutki, millaisen virheilmoituksen Python-tulkki ilmoittaa. Aloita uusi tiedosto ja kirjoita koodi

 # syntaksivirheitä
***prin** ("Väärä komento")*

Tallenna koodi KOODIT kansioon nimellä *virheet*. Suorita ohjelma. Mikä virheilmoitus tulostuu?

_____

Korjaa print-komento ja poista toinen lainausmerkki

 # syntaksivirheitä
***print** ("Väärä komento)*

Virheilmoitus:_____

Korjaa virhe ja tallenna koodi uudelleen. Koodi-ikkunoita voi olla avoinna useita. Tarpeettomia ikkunoita on kuitenkin hyvä sulkea.

Tehtävä 10. Tulostuksen rivittäminen

Koodaa ohjelma, joka tulostaa oheisen mallin mukaisen tulosteen näytölle. Käytä print-komennossa **kolmoislainausmerkkejä**, niin saat tulostukseen näkyviin tekstien rivinvaihdot ja sisennykset yhdellä print-komennolla.

Lukujoukkoja:
Luonnolliset luvut: N= {0, 1, 2, 3, ...}
Kokonaisluvut: Z= {-3, -2, -1, 0, 1, 2, 3, ...}
Rationaaliluvut: Q= {Z + murtoluvut}

Tallenna koodi KOODIT-kansioon nimellä *lukujoukot*.

Suorita ohjelma.

3. Merkkijono, lukujen ja laskulausekkeiden tulostaminen

Opit uutta:
- merkkijono tietotyyppi
- kirjoittamaan laskulausekkeita
- tulostamaan merkkijonon ja laskulausekkeen yhdessä

Osaat jo:
- print-komennon
- tallentaa koodin haluamaasi kansioon
- kirjoittaa koodiin kommentteja

Print-komennolla saadaan tulostettua näytölle mikä tahansa tieto, kunhan komento on kokonaisuudessaan oikein kirjoitettu (syntaksin mukaisesti). Print-komennon sulkeisiin merkkijonotyyppinen tieto kirjoitetaan lainausmerkkien sisään. **Merkkijono eli string** on ohjelmoinnissa yksi tietotyyppi. Merkkijono voi olla tyhjä, jolloin siinä on vain lainausmerkit (””) tai se voi sisältää useita merkkejä.

Myös merkkijonossa lainausmerkkien sisään kirjoitetut luvut ovat merkkityyppisiä ja tulostuvat kirjoitetussa muodossaan.

```
print (””)
# tyhjä merkkijono
```

```
print (’Summa 1+2’)
```

Summa 1+2

```
print (’’’ Mikä on lausekkeen tulos
2*4+7*10 = ’’’)
```

Mikä on lausekkeen tulos
2*4-7*10 =

Muistathan, että kolmoislainausmerkeillä saadaan merkkijono rivitettyä.

Pythonissa tähtimerkki (*) on kertomerkki. Esim. 2*4 on kaksi kertaa neljä.

Jos laskulausekkeiden tulostamisessa halutaan saada näkyviin lausekkeen arvo, kirjoitetaan laskulauseke ilman lainausmerkkejä. Tällöin Python-tulkki suorittaa laskutoimituksen ennen tulostamista ja lausekkeen paikalla tulostuu laskulausekkeen arvo. Myös **lukutyyppisen** tiedon (**kokonaisluvut** ja **desimaaliluvut**) tulostamisessa jätetään lainausmerkit pois. Jos samaan tulostukseen yhdistetään merkkijonoja ja laskulausekkeita, ne tulee erottaa pilkulla.

```
print (2*3+4)
# ensin laskutoimitus, sitten tulostus
```

10

```
print (”Kokonaisluku”, 12)
print (”Desimaaliluku”, 5.75)
print (”Lukujen 12 ja 5.75 summa on”, 12+5.75)
```

Kokonaisluku 12
Desimaaliluku 5.75
Lukujen 12 ja 5.75 summa on 17.75

Tulostettaessa samassa print-lauseessa sekä merkkijono- että lukutyyppejä, erotetaan ne pilkulla

Huomaa, että desimaaliluvussa erottimena on **piste**, ei pilkku

```
print (”Lukujen 2 ja 10 summa on”,2+10,” ja tulo on”,2*10)
```

Lukujen 2 ja 10 summa on 12 ja tulo on 20

Tulostuslauseen pilkkujen ja lainausmerkkien kanssa tulee olla huolellinen. Yksikin väärässä kohdassa oleva merkki aiheuttaa Python-tulkin virheilmoituksen.

TEHTÄVIÄ

Tehtävä 11. Lukujen ja laskulausekkeiden tulostaminen

Tutki, mitä seuraavat print-lauseet tulostavat. Päättele ensin ja kokeile sitten koodaamalla rivit Pythonilla.

print ("Lukujen -7 ja -6 tulo on:", -7(-6))*

tulostus: _____

*print (1*2*3*4+6)*

tulostus: _____

*print(1+2+3+4*6)*

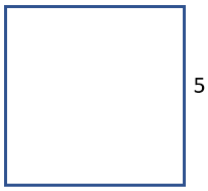
tulostus: _____

Tehtävä 12. Neliön piiri ja pinta-ala

Koodaa ohjelma, joka laskee oheisen neliön **piirin** ja **pinta-alan.** Ohjelma tulostaa piirin ja pinta-alan. Yhdistä siis tulostuslauseessa merkkijonon loppuun laskulausekkeet; esimerkiksi

*print ("Neliön pinta-ala on", 5*5)*

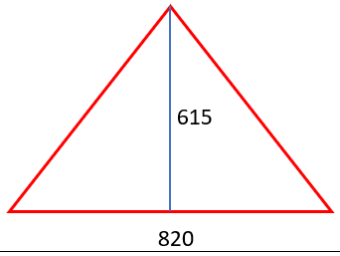
Tallenna ohjelma KOODIT-kansioosi nimellä *neliö* ja testaa ohjelman toiminta.



Tehtävä 13. Kolmion pinta-ala

Koodaa ohjelma, joka laskee oheisen kolmion **pinta-alan.** Kuvassa on annettu kolmion kannan pituus ja korkeus. Ohjelma myös tulostaa pinta-alan. Yhdistä siis tulostuslauseessa merkkijonon loppuun laskulauseke.

Tallenna ohjelma KOODIT-kansioosi nimellä *kolmio* ja testaa ohjelman toiminta



Tehtävä 14. Simon ensimmäinen ohjelma

Simo on yrittänyt koodata yksinkertaisen ohjelman, mutta koodissa on useita virheitä. Koodissa on sekä syntaksi- että laskennallisia virheitä. Kirjoita koodi syntaktisesti oikein ja korjaa myös matemaattiset virheet. Tallenna korjattu ohjelma KOODIT-kansioosi nimellä *omenaostos.*

@ matemaattinen tehtävä
print "Omenat maksavat 2 euroa/kg)
print Matti ostaa omenoita 2,5 kg"
print ("Omenaostos maksaa silloin: 2,5+2)

4. Peruslaskutoimituksia Pythonilla, aritmeettiset operaattorit

Opit uutta:

- kokonaisluku tietotyyppi eli integer
- desimaaliluku tietotyyppi eli float
- totuusarvo tietotyyppi eli bool
- aritmeettiset operaattorit ja operaatiot: summa, erotus, osamäärä, tasajako, jakojäännös, potenssi, sulkeiden käyttö lausekkeissa

Osaat jo:

- tulostaa merkkijonoja ja laskulausekkeita
- kirjoittaa koodin matemaattisesta (geometrisesta) mallista
- tunnet tietotyypin merkkijono

Pythonissa aritmeettiset lausekkeet eli **operaatiot** suoritetaan samalla tavalla kuin matematiikassakin. Ensinnä lasketaan sulkeiden sisällä olevat laskutoimitukset, seuraavaksi potenssit, sitten kerto- ja jakolaskut vasemmalta oikealle ja lopuksi yhteen- ja vähennyslaskut. Laskulausekkeissa käytettyjä lukuja ja lukumuuttujia kutsutaan operandeiksi ja laskutoimitusmerkkejä operaattoreiksi. Taulukossa on lueteltu aritmeettiset operaatiot.

Operaatio	Operaation tulos
a + b	Lukujen a ja b summa.
a – b	Lukujen a ja b erotus.
–a	Luvun a etumerkin vaihto eli luvun a vastaluku.
a * b	Lukujen a ja b tulo.
a / b	Lukujen a ja b osamäärä.
a // b	Tasajako eli lukujen a ja b jakolaskun kokonaisosa.
a % b	Lukujen a ja b osamäärän jakojäännös.
a**b	Luku a korotetaan potenssiin b.

Osaatko laskea lausekkeet oikein? Käytä oheista operaatiotaulukkoa apuna ja kirjoita lausekkeiden vastaukset. Luvun a arvo on 7 ja luvun b arvo on 2.
a = 7
b = 2

–a =_____

a * b = _____

a**//**b = _____

a **%** b + a **%** b = _____

a******b =_____

(a**//**b) * (a**%**b) = _____

(a******b) **/** (a****2**) =_____

(a+b) * (a–b) =_____

Huom! Voit laskea lausekkeiden tulokset myös kirjoittamalla ne Pythonin **Shell**-ikkunaan. Kirjoita ensin *a=7* ja *b=2*. Sen jälkeen voit kirjoittaa lausekkeet kirjaimien avulla. Älä kuitenkaan kirjoita lausekkeiden loppuun '=' merkkiä. Sille on Pythonissa eri merkitys kuin matematiikassa. Tämä selviää tehtävässä no 5.

Perustietotyyppi	Python-kielen sana	Arvoalue	Esimerkki arvon asetuksesta
totuusarvot	bool	True tai False 0 tai 1	a = True, b = False (True ja False kirjoitetaan isolla alkukirjaimella.)
kokonaisluvut	int	Rajaton, mutta tietokoneen muisti rajoittaa käytettävää lukualuetta.	n = 125 i = –2000
desimaaliluvut	float	Arvot korkeintaan $1.79*10^{308}$, vähintään $5.0*10^{-324}$, mutta riippuu järjestelmän ominaisuuksista.	x = 2.45 y = –0.0075 (Desimaalierottimena käytetään pistettä.)

Matemaattisten ohjelmien yhteydessä tärkeimmät lukutyypit ovat **kokonaisluku** (integer, int), **desimaaliluku** (float) ja **totuusarvo** (boolean, bool).

Totuusarvot ovat kaksiarvoisia lukuja (0/1, True/False), ja ne saavat arvon *tosi* tai *epätosi*. Totuusarvoja käytetään ohjelman haarautumiseen ehto- ja toistolauseissa.

Huomaa, että desimaaliluvussa erottimena on **piste**, ei pilkku.

```
print (3.14)
```

```
3.14
```

```
print (2*(3+4) / 7-0)
# sulkeilla saadaan muutettua laskujärjestystä
```

```
0
```

Onkohan vastaus oikein?

Tarkista koodi "pöytätestaamalla" eli laskemalla se itse.

TEHTÄVIÄ

Tehtävä 15. Merkkijono ja laskulauseke tulostuksessa

Täydennä seuraava ohjelma siten, että se tulostaa mainitun matemaattisen lausekkeen arvon. Päättele ensin ja koodaa sitten merkkijonojen perään laskutoimituksen suorittava lauseke. Tallenna valmis ohjelma KOODIT-kansioosi nimellä *aritmetiikkaa*.

print (”Lukujen 25 ja -12 vastaluvut ovat”, ,)

print (”Lukujen 6 ja 5 summan ja erotuksen tulo on” ,)

print (”Lukujen 15 ja -12 tasajako antaa tuloksen”,)

print (”Jakolaskun 3 jaettuna 8:lla jakojäännös on”,)

Tehtävä 16. Desimaalipiste liikkuu

Desimaalilukua kerrottaessa ja jaettaessa 10:llä, 100:lla tai vaikkapa 1000:lla, siirtyy desimaalipilkku. Mutta miten se siirtyy? Voit varmistaa tietosi seuraavalla ohjelmalla. Koodaa siis ohjelma ja tallenna KOODIT-kansioon nimellä *kertaluokka*. Suorita ohjelma ja kirjoita ’pilkkusääntö’ omin sanoin havaintosi perusteella.

 # desimaaliluvun kertominen ja jakaminen 10, 100, 1000
*print (3.14 * 10)*
*print (3.14 * 100)*
*print (3.14 * 1000)*

print (314.0 / 10)
print (314.0 / 100)
print (314.0 / 1000)

SÄÄNTÖ: _____

_____

Tehtävä 17. Yksinkertaista Simon lausekkeet

Seuraavassa koodissa Simo on kirjoittanut laskulausekkeet tarpeettoman pitkästi. Tehtäväsi on kirjoittaa lausekkeet lyhyemmin (sievennettynä). Käytä sieventämisessä aritmeettisia operaattoreita. Tallenna yksinkertaistettu ohjelma KOODIT-kansioon nimellä *sievennykset*.

 # lausekkeiden sievennys
print (2+2+2+2+2+2+2+2+2+2)
print (-5+(-5)+(-5)+(-5)+(-5)+(-5)+(-5)+(-5))
print (4+(-10)+4+(-10)+4+(-10)+4+(-10)+4+(-10)+4+(-10))
*print (7*7*7*7*7*7*7*7*7*7)*
*print ((-5+(-5)+(-5)+(-5))+7*7*7*7*7)*

Tehtävä 18. Korjaa Simon tulostuslauseet

Simo on yrittänyt koodata tulostuslauseita, mutta koodissa on useita virheitä. Koodissa on sekä syntaksi- että laskennallisia virheitä. Kirjoita koodi syntaktisesti oikein ja korjaa myös matemaattiset virheet. Tallenna korjattu ohjelma KOODIT-kansioosi nimellä *laskuvirheet.*

print (”Desimaalilukujen summa’, 12,45 + 10,55

print (”Kolme potenssiin neljä on” 3^4)

print (Jakojäännös osamäärästä 12÷5 on”, 12&5)

print (Lukujen 5,5 ja 12,5 tulo on”, 5,5 x 12,5)

5. Muuttujat, arvon asettaminen muuttujaan asetuslauseella

Opit uutta:
- muuttujan nimeäminen
- merkkijonomuuttujan arvon asetus ja arvon muuttaminen
- merkkijonojen yhdistäminen + merkillä

Osaat jo:
- merkkijono- ja lukutyypit
- laatia aritmeettisia laskuoperaatioita ohjelmoimalla
- tulostaa merkkijonoja

Muuttujia tarvitaan ohjelman suorittamiseen ja tietojen väliaikaiseen varastointiin. Muuttujalle varataan automaattisesti tietokoneen muistista 'säilytystila', johon arvo tallennetaan. Muuttujalle annetaan **tunnisteeksi** sen tarkoitusta kuvaava nimi. Ristiriitojen välttämiseksi muuttujat eivät saa olla keskenään samannimisiä. Muuttujan nimeämiselle on seuraavat säännöt:

Sääntö	**Kelvollisia tunnisteita**	**Kelvottomia tunnisteita**
Pitää alkaa kirjaimella. Ei saa olla väliviivaa, mutta alaviiva saa olla.	etunimi toinen_nimi	1nimi toka-nimi
Ei saa sisältää erikoismerkkejä.	pelaaja_nro dollarit ale_pros	#pelaaja dollar$ ale%
Ei saa olla välilyöntejä.	rekisteri_numero	auton rekisterinumero

Python erottelee isot ja pienet kirjaimet. Esimerkiksi muuttujan nimet *osoite* ja *Osoite* ovat eri muuttujia. Skandinaavisten merkkien (ääöÄÄÖ) käyttöä on yhteensopimattomuussyistä vältettävä.

```
nimi = ”Senni”
joukkue = ”Salamat”
rooli = ”puolustaja”

print (”Pelaaja:”, nimi)
print (”Joukkue:”, joukkue, ”Pelipaikka: ”,rooli)
```

```
Pelaaja: Senni
Joukkue: Salamat Pelipaikka: puolustaja
```

Samassa tulostuslauseessa voidaan tulostaa merkkijonoja ja muuttujien arvoja, kunhan ne erotetaan pilkulla.

```
print (”Pelaaja:”, nimi)
```

merkkijono — muuttuja

Pilkku erottaa eri tyyppiset tulostettavat

Muuttujan arvo asetetaan ja muutetaan **asetuslauseella**. Pythonissa asetusoperaattorina eli asetuksen suoritusmerkkinä käytetään **=** -merkkiä. Tätä ei pidä sekoittaa matematiikan yhtäsuuruusmerkkiin. **Asetuslauseessa = merkin oikealla puolella oleva arvo asetetaan vasemmalla puolella olevan tunnisteen arvoksi**.
Esimerkiksi asetuslause

osoite = ”Rantakatu 3”

asettaa *osoite* -muuttujan arvoksi merkkijonon *Rantakatu 3*. Jos osoite -muuttujan arvo halutaan tulostaa, se onnistuu print -komennolla

print (osoite)

Huomaa, että muuttujan arvoa tulostettaessa, sitä ei kirjoiteta lainausmerkkien sisään.

```
merkkijono = ”Aurinko mailleen vajoaa”
print (merkkijono)
merkkijono = ”Lintujen laulu vaimenee”
print (merkkijono)
```

```
Aurinko mailleen vajoaa
Lintujen laulu vaimenee
```

Ohjelmakoodissa muuttujan arvoa voidaan muuttaa. Uudelleenasetus tehdään kirjoittamalla samalle tunnisteelle uusi asetuslause. Muutoksen jälkeen vanha arvo katoaa, eikä sitä ole mahdollista palauttaa ilman uudelleen asetusta.

Oheisen runon lauseet tulee tallentaa erinimisiin muuttujiin. Tämän jälkeen ne on mahdollista tulostaa yhdelle riville. Merkkijonoja voidaan yhdistää (katenoimalla) **+** merkillä. Esimerkiksi: asetetaan muuttujat *a, b* ja *c* merkkijonoarvoilla ja tulostetaan arvot yhdessä print-lauseessa.

```
a = ”viisas ”
b = ”pääsee ”
c = ”vähemmällä”
print (a+b+c)
```

```
viisas pääsee vähemmällä
```

Huomaa, että kahden ensimmäisen merkkijonon perään on lisätty välilyönti. Ilman tätä katenointi yhdistäisi merkkijonot ilman välilyöntiä *viisaspääseevähemmällä*.

TEHTÄVIÄ

Tehtävä 19. Merkkijonomuuttujia

Täydennä alla oleva koodi. Aseta muuttujien arvoiksi omat tietosi. Täydennä myös print-lauseet sellaisiksi, että saat tulostettua muuttujien arvot, kukin omalle rivilleen.

etunimi =
tokanimi =
sukunimi =

print (”Etunimi:”,
print (”Toinen nimi:”,
print (”Sukunimi:”,

Tallenna täydennetty ohjelma KOODIT-kansioon nimellä *merkkimuuttujat*. Testaa ohjelman toiminta.

Tehtävä 20. Merkkijonojen yhdistäminen eli katenointi

Merkkityyppisiä muuttujia voidaan tulostuslauseessa yhdistää + merkillä. Toisaalta tulostuslauseessa voidaan tulostettavat muuttujat luetella pilkuila eroteltuna. Kirjoita seuraava koodi ja tutki, miten tulostustavat eroavat toisistaan lopputuloksen kannalta?

s = "Suurin"
y = "yhteinen"
t = "tekijä"
print (s+y+t)
print (s,y,t)

Tallenna ohjelma KOODIT-kansioon nimellä *katenointi*.

TULOSTUKSET EROAVAT _____

_____

LISÄTEHTÄVÄ: yhdistä tulostuslauseeseen *print (s+y+t)* välilyönnit (” ”) muuttujien väliin, jotta myös tämän rivin tulostus saadaan oikein välilyöntien kanssa.

Tehtävä 21. Merkkien monistaminen

Merkkejä ja merkkijonoja voidaan Pythonissa monistaa * operaattorilla. Esimerkiksi lause *print (5*”OHO”)* tulostaa OHOOHOOHOOHOOHO.

Mieti, miten saat tulostettua oheisen kuvion käyttämällä välilyönti (” ”) ja ”#” merkkiä sekä merkkien monistusta. Tallenna ohjelma KOODIT-kansioon nimellä *pyramidi.* Testaa tulostus.

```
#
###
#####
```

Tehtävä 22. Simon merkkijonon monistus

Simo on yrittänyt koodata merkkejä toistavan ohjelman, mutta koodissa on paljon syntaksivirheitä ja virheellisiä tunnisteita. Korjaa virheet, tallenna koodi ja suorita ohjelma. Mitä koodi tulostaa?

”papu” = a
”pata” = b
$toisto kerrat = 5
print (”Tulostetaan useita kertoja”, a+b)
*print (toistokerrat * a+b)*

Oikean tulosteen pitäisi näyttää seuraavalta:

```
Tulostetaan useita kertoja papupata
papupatapapupatapapupatapapupatapapupata
```

6. Näppäinsyöte, input, tyypinmuunnos kokonaisluvuksi

Opit uutta:
- näppäinsyöte input-komennolla
- kokonaislukumuuttujan arvon asettaminen näppäinsyötteestä
- tyypinmuunnos kokonaisluvuksi

Osaat jo:
- asetuslauseen
- vieruskulman ja ristikulman käsitteet geometriasta
- tulostaa muuttujien arvoja
- yhdistää merkkijonoja

Käyttäjä voi syöttää ohjelmaan tietoja **input**-komennolla. Näppäimistöllä syötetty tieto on aina merkkityyppistä (string), eikä se kelpaa esimerkiksi laskutoimituksissa käytettäväksi. Laskutoimituksia varten näppäinsyöte pitää muuntaa **lukutyyppiseksi**. Helpointa muunnos on tehdä yhdistämällä tyypinmuunnos ja input-lause.

input-komennossa sulkeisiin kirjoitetaan näytettävä teksti. Tekstissä kerrotaan, mitä käyttäjän tulee kirjoittaa.

```
# luetaan näppäimistöltä
etunimi = input(”Mikä on etunimesi? ”)
sukunimi = input(”Entä sukunimesi? ”)
print (etunimi, sukunimi)
```

```
Mikä on etunimesi? Maija
Entä sukunimesi? Salonen
Maija Salonen
```

Jokaisen input-komennon suorituksessa ohjelma pysähtyy odottamaan käyttäjän antamaa syötettä. Ohjelman suoritus jatkuu enter-näppäimen painalluksen jälkeen.

1. Käyttäjälle tulostettava kehote. Sulkeita on kahdet.
2. Luetaan näppäinsyöte tietokoneen muistiin.
3. Muunnetaan syötteen tyyppi, tässä desimaaliluvuksi.

```
luku = int(input(”Anna jokin luku: ”))
print (”Annoit luvun”, luku)
```

```
Anna jokin luku: 175
Annoit luvun 175
```

Näppäinsyöte tallennetaan muuttujaan **asetuslauseella**. Jos tallennettavana on lukutyyppinen tieto, on näppäinsyöte muunnettava lukutyypiksi. Muunnos esimerkiksi **kokonaisluvuksi** (int) tehdään kirjoittamalla input-komento sulkeisiin ja sulkeiden eteen kirjoitetaan kokonaisluvun tyyppitunnus *int*. Esimerkiksi lause

luku = int(input(”Anna jokin luku”: ”))

lukee näppäimistösyötteen *input*-komennolla ja muuntaa syötteen *int*-tyyppiseksi. Asetusoperaattorilla syöte asetetaan *luku*-muuttujan arvoksi.

```
# luetaan kokonaislukuja ja lasketaan summa
a = int(input(”Anna luku a: ”))
b = int(input(”Anna luku b: ”))
summa = a + b
print (”Lukujen”,a,”ja”,b,”summa on”,summa)
```

```
Anna luku a: 34
Anna luku b: 56
Lukujen 34 ja 56 summa on 90
```

input-komennon kehotetekstin voi päättää välilyöntimerkkiin, jolloin tekstikohdistin ei tulostu heti tekstin perään ilman väliä.

Kun tulostuslauseessa yhdistetään muuttujia ja merkkijonoja, on pilkkujen ja lainausmerkkien kanssa oltava erityisen tarkkana.

TEHTÄVIÄ

Tehtävä 23. Tietoa näppäimistöltä

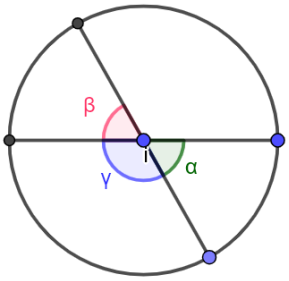
Edellisen sivulla on esimerkki, jossa luetaan kaksikokonaislukua a ja b sekä tulostetaan näiden summa. Koodaa ohjelma ja lisää siihen vielä lukujen a ja b aritmeettiset operaatiot *erotus, tulo, osamäärä, tasajako* ja *jakojäännös.*

Operaatioiden tulos voidaan asettaa muuttujaan tai laskulauseke voidaan yhdistää tulostuslauseeseen. Ohjelman pitäisi tulostaa seuraavanlaisesti:

Anna luku a: ***13***
Anna luku b: ***9***
Lukujen 13 ja 9 summa on 22
Lukujen 13 ja 9 erotus on 4
Lukujen 13 ja 9 tulo on 117
Lukujen 13 ja 9 osamäärä on 1.4444444444444444
Lukujen 13 ja 9 tasajako on 1
Lukujen 13 ja 9 jakojäännös on 4

Tehtävä 24. Keskuskulma, ristikulma, vieruskulma

Ympyrään on piirretty kaksi halkaisijaa kuvan osoittamalla tavalla. Laadi ohjelma, joka laskee kulmien β (*beta*) ja γ (*gamma*)suuruudet, kun ohjelmaan syötetään kulman α (*alfa*) suuruus. Ohjelma kysyy kulman α ja laskee tämän avulla muut kulmat. Kulmien arvo tulostetaan yhdellä *print*-lauseella.



Tallenna ohjelma KOODIT-kansioon nimellä *keskuskulmat*. Varmista ohjelman toiminta.

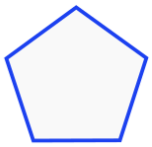
Tehtävä 25. Säännöllisen monikulmion kulma

Säännöllisen monikulmion kulman α suuruus saadaan laskettua kaavalla

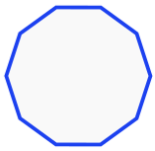
$\alpha = \frac{n-2}{n} \cdot 180°$, missä n on monikulmion kulmien määrä

Esimerkiksi neliön kulman suuruus on

$$\alpha = \frac{4-2}{4} \cdot 180° = 90°$$



n = 5



n = 10

Koodaa ohjelma, joka kysyy käyttäjältä säännöllisen monikulmion kulmien lukumäärän *n* ja laskee monikulmion kulman α suuruuden.

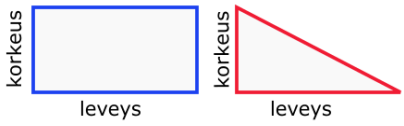
Ohjelma tulostaa kulman suuruuden ja lisäksi tulostetaan monikulmion kaikkien kulmien summa.

Tallenna ohjelma KOODIT-kansioon nimellä *monikulmio*. Testaa ohjelman toiminta

Tehtävä 26. Simon suorakulmio ja kolmio

Simo osaa hyvin geometriaa, mutta ohjelmoinnissa on vielä harjoiteltavaa. Simo on kirjoittanut oheisen koodin, mutta siinä on useita huolimattomuusvirheitä. Korjaa virheet ja tallenna korjattu ohjelma KOODIT-kansioon nimellä *suorakulmion_ja_kolmion_ala*. Leveys ja korkeus annetaan kokonaislukuina.

```
# lasketaan suorakulmion ja
# vastaavilla sivujen pituuksilla esitetyn suorakulmaisen kolmion alat
leveys = keyboard("Anna leveys kokonaislukuna")
korkeus = keyboard("Anna korkeus kokonaislukuna")

leveys*korkeus = sk_ala  # suorakulmion ala

leveys*korkeus/2 = kolmion_ala  # vastaavan suorakulmaisen kolmion ala

print "Suorakulmion ala on" sk_ala "ja kolmion ala on " kolmion_ala"
```



7. Muunnos desimaaliluvuksi, pyöristäminen, yhdistetyt operaatiot

Opit uutta:
- tyypinmuunnos desimaaliluvuksi
- yhdistetyt operaatiot
- desimaaliluvun pyöristäminen round-komennolla

Osaat jo:
- tyypinmuunnoksen kokonaisluvuksi
- input-komennon
- merkkijonojen ja muuttujien tulostaminen

Tyypinmuunnoksissa on muunnettavan syöttötiedon oltava kelvollinen, jotta se voidaan muuntaa esimerkiksi lukutyyppiseksi. Esimerkiksi merkkijonoa ”*kissa*” ei voida muuntaa luvuksi, vaan Python tulkki antaa virheilmoituksen. Kun näppäimistösyöte halutaan muuntaa desimaaliluvuksi (float), pitää input -komennon eteen kirjoittaa lukutyyppi *float* ja input-komento kirjoitetaan sulkeisiin.

```
# luetaan näppäimistöltä desimaaliluku
luku = float(input(”Anna desimaaliluku:  ”))
print (”Syötit luvun”, luku)
```

```
Anna desimaaliluku: 67.342
Syötit luvun: 67.342
```

Näppäinsyöte saadaan muunnettua desimaalityyppiseksi ’kastamalla’ syöte **float**-tyyppiseksi. Syötteen tulee olla desimaaliluvuksi kelpaavassa muodossa, eli desimaalierottimena käytetään pistettä.

Tyypinmuunnos float-tyypiksi on parempi verrattuna int-tyypiksi. Float-tyyppisiä lukuja voidaan laskea samoin kuin int-tyyppisiä, mutta käyttäjällä on laajempi lukualue käytössään. .

```
# yhdistetyt asetusoperaatiot
a = b = c = d = 2
a += 100 # a:han lisätään luku 100
b –= 30  # b:stä vähennetään 30
c **= 6  # c korotetaan potenssiin 6
d %= 2   # d:n ja luvun 2 jakojäännös
f *= 10  # f muuttuja 10-kertaisena
g /= 2   # g muuttuja jaetaan kahdella
print ("Tulokset:",a,b,c,d)
```

```
Tulokset: 102 -28 64 0 20 1.0
```

Lause *a = b = c = d = 2* alustaa muuttujien *a,b,c* ja *d* arvoksi luvun 2 samanaikaisesti.

Lukumuuttujien arvoa voidaan muuttaa **yhdistetyllä operaatiolla**. Yhdistetyssä operaatiossa tehdään muuttujalle jokin aritmeettinen operaatio ja samassa lauseessa arvo asetetaan muuttujaan. Seuraavassa koodissa on esimerkkejä yhdistetyistä operaatioista kommentteineen.’

Yhdistettyjä operaatioita ei pidetä hyvän ohjelmointitavan mukaisina, mutta ne lyhentävät koodin pituutta jonkin verran. Siksi niitä käytetään eri yhteyksissä ja siksi ne on hyvä tuntea.

Desimaalilukuja pyöristetään round() -komennolla.

Desimaalilukuja voidaan pyöristää ***round(x,n)***-komennolla, jossa *x* on jokin float-tyyppinen luku ja *n* on esitettävien desimaalien määrä. Esimerkiksi *round*(2.4567,2*)* antaa tulokseksi 2.46. Jos desimaalien määrää ei anneta, pyöristetään luku kokonaisluvuksi, esim. *round(2.4567)* pyöristetään arvoon 2.

Pyöristys poikkeaa matematiikasta tutusta pyöristyssäännöstä, jossa ensimmäisen poisjäävän numeron ollessa suurempi tai yhtä suuri kuin 5, niin luku pyöristetään ylöspäin. Esimerkiksi perusmatematiikan mukaan luku 3.65 on yhden desimaalin tarkkuuteen pyöristettynä 3.7. Python pyöristäisi saman luvun arvoon 3.6. Huomaa, että ohjelmoinnissa desimaalierottimena käytetään pistettä.

Pythonin versiossa 3 *round()*-komento pyöristää desimaaliluvun niin sanotun **pankkiirin pyöristyssäännön** mukaan. Pankkiirin pyöristyssääntö on seuraava: jos desimaaliluvun pyöristyksen virhe on yhtä suuri ylös- tai alaspäin, luku pyöristetään parilliseen numeroon päättyvään lukuun. Esimerkiksi luku 0.5 pyöristetään kokonaisluvuksi 0. Luku 1.5 pyöristetään kokonaisluvuksi 2. Desimaaliluku 27.25 olisi yhden desimaalin tarkkuudella 25.2. Laskutoimituksissa on hyvä muistaa, että vain lopputulos pyöristetään. Jos pyöristyksiä tehdään välivaiheisiin, tulee vastaukseen liian suuri virhe.

```
# pyöristystä
luku =float(input("Anna luku, jossa on viisi desimaalia: "))
print ("Tuhannesosan tarkkuudella",round(luku,3))
print ("Sadasosan tarkkuudella",round(luku,2))
print ("Kymmenesosan tarkkuudella",round(luku,1))
```

```
Anna luku, jossa on viisi desimaalia: 1.55555
Tuhannesosan tarkkuudella 1.556
Sadasosan tarkkuudella 1.56
Kymmenesosan tarkkuudella 1.6
```

TEHTÄVIÄ

Tehtävä 27. Float-lukujen pyöristäminen

Desimaaliluvut (float) pyöristetään *round*-komennolla. Lisää seuraaviin tulostuslauseisiin *round*-pyöristykset siten, että luku pyöristetään ja tulostetaan mainittuun tarkkuuteen.

täydennä puuttuvat pyöristykset

```
# pyöristystä
luku = 10.76547
print ("Tuhannesosan tarkkuudella",round(     ))
print ("Kahden desimaalin tarkkuudella",round(     ))
print ("Kymmenestuhannesosan tarkkuudella",round(     ))
print ("Kokonaislukuna",round(     ))
```

Tallenna täydennetty ohjelma KOODIT-kansioosi nimellä *pyöristystä* ja testaa ohjelman toiminta.

Tehtävä 28. Desimaalilukujen kerto- ja jakolaskuja

Koodaa ohjelma, joka kysyy kaksi desimaalilukua ja tulostaa näiden tulon ja osamäärän kahden desimaalin tarkkuudella. Ohjelman toiminta voisi näyttää seuraavalta.

```
Anna a: 0.0745
Anna b: 1.234
0.0745 * 1.234 = 0.09
0.0745 ÷ 1.234 = 0.06
```

Koodaa tulostuslauseet mallin mukaisesti. Tallenna toimiva ohjelma KOODIT-kansioon.

Laske ohjelmaasi käyttämällä seuraavat kerto- ja jakolaskut:

9.999 · 0.9999 = _____ 9.999 ÷ 0.9999 = _____

333.33 · 0.33333 = _____ 333.33 ÷ 0.33333 = _____

Tehtävä 29. Kertolasku potenssiksi ja summa kertolaskuksi

Simolla on vielä harjoiteltavaa koodin tehokkaassa kirjoittamisessa. Alla on Simon kirjoittama koodi, joka laskee tietyt operaatiot luvuille a ja b. Sievennä laskulausekkeet siten, että molempien muuttujien operaatiot mahtuvat yhdelle riville. Tallenna sievennetty ohjelma KOODIT-kansioon nimellä *lyhyet*.

```
# sievennä koodi ja merkitse yhdistetyt operaatiot lyhyemmin
a = float(input("Anna luku a: "))
a *= 10
a *= 10
a *= 10
a *= 10
a *= 10
print (a)
b = float(input("Anna luku b: "))
b += 100
b += 100
b += 100
b += 100
b += 100
b += 100
b += 100
b += 100
b += 100
b += 100
b += 100
print (b)
```

Tehtävä 30. Pankkiirinpyöristys

Osaatko selittää pankkiirin pyöristyssäännön esimerkiksi lukujen 1.455 ja 1.45 osalta, kun molemmat luvut pyöristetään yhden desimaalin tarkkuuteen?

Koodaa ohjelma, joka kysyy kaksi float-lukua. Ohjelman tulee pyöristää ja tulostaa luvut yhden desimaalin tarkkuuteen. Syötä luvut 1.455 ja 1.45.
Miten lukujen pyöristetyt tulokset eroavat?

Luku 1.455 on Pythonissa pyöristettynä yhden desimaalin tarkkuuteen ____.

Luku 1.45 on Pythonissa pyöristettynä yhden desimaalin tarkkuuteen ____.

Miten selität pyöristysten erilaiset lopputulokset pankkiirin pyöristyssäännön perusteella?

_____

_____

_____

8. Matematiikkaa Pythonilla, math-kirjasto

Opit uutta:

- math-kirjaston käytön matemaattisissa sovelluksissa
- itseisarvo, abs
- potenssi, pow

Osaat jo:

- Python-ohjelmoinnin perusteita:

merkkijonot, lukutyypit, tyypinmuunnos, asetuslauseen, muuttujan arvon asetus, tulostaa merkkijonoja ja muuttujien arvoja, pyöristää desimaalilukuja

Pythonissa on lukuisia **kirjastomoduuleja**, joiden avulla ohjelmiin saadaan hyödyllisiä toimintoja. Moduulit otetaan käyttöön *import*-lauseella. ***Import*-lause** kirjoitetaan ohjelmaan ensimmäiseksi riviksi. Kirjastomoduulin kaikki toiminnot saadaan käyttöön kirjoittamalla *import*-lause muodossa ***from moduuli import ** (*-merkki tarkoittaa kaikkia toimintoja). Esimerkiksi matemaattisissa sovelluksissa tarvitaan usein *math*-kirjastoa, jolloin ohjelman ensimmäiseksi riviksi tulee kirjoittaa *from math import **. Seuraavassa taulukossa on peruskoulun matematiikassa tarvittavia *math*-kirjaston toimintoja.

Math-kirjaston toiminto	**Toiminnon kuvaus**
fabs(*x*)	Palauttaa luvun *x* itseisarvon (*englanniksi absolute value*). Palautusarvo on float-tyyppinen.
gcd(*a*,*b*)	Palauttaa lukujen *a* ja *b* suurimman yhteisen tekijän (*englanniksi greatest common divisor*).
pow(*x*,*y*)	Palauttaa x^y. Sama kuin *x**y*. (*englanniksi power*).
sqrt(*x*)	Palauttaa neliöjuuren luvusta *x* (*englanniksi square root*).

matematiikkakirjastosta math — otetaan käyttöön, — *from math import ** — * merkki tarkoittaa 'kaikki komennot'

Pythonissa luvun itseisarvo saadaan komennolla ***abs(x)*** sekä math-kirjaston ***fabs(x)*** komennolla. Tutustumme math-kirjaston käyttämiseen, joten käytetään tässä tehtävässä *fabs()*-komentoa. Esimerkiksi *fabs(-8)* antaa arvon 8.0. Vastaavasti lauseessa *fabs(-5-3)* lasketaan ensin sulkeissa oleva -5 - 3 = -8 ja sen jälkeen saadaan luvun -8 itseisarvo 8.0.

otetaan käyttöön math-kirjasto

```
# math-kirjastoesimerkki
from math import *
a = 12
b = 9
c = -120
print ("Luvun",c,"itseisarvo on",fabs(c))
print ("Lukujen",a,"ja",b,"suurin yhteinen tekijä on",gcd(a,b))
print (a,"potenssiin",b,"on",pow(a,b))
print ("Neliöjuuri luvusta",b,"on",sqrt(b))
```

```
Luvun -120 itseisarvo on 120.0
Lukujen 12 ja 9 suurin yhteinen tekijä on 3
12 potenssiin 9 on 5159780352.0
Neliöjuuri luvusta 9 on 3.0
```

$|x|$ **Itseisarvo |x|** on luvun x etäisyys 0:sta. Kahden luvun suuruusero saadaan lukujen erotuksen itseisarvona. Esimerkiksi lukujen -5 ja 3 suuruusero on **|-5-3|** = 8 .

$syt(a,b)$ Kahden kokonaisluvun **suurin yhteinen tekijä** eli syt on luku, jolla molemmat luvut voidaan jakaa pienimpään kokonaislukumuotoonsa. Jos a ja b eivät ole keskenään jaollisia, niin silloin syt(a,b) = 1. Pythonissa *syt*-laskutoimitusta vastaa *gdc()*.

$\sqrt{x}$ Luvun x **neliöjuuri** $\sqrt{x}$on sellainen positiivinen luku, joka kerrottuna itsellään on x. Eli $\sqrt{x} \cdot \sqrt{x}$= x. Esim. $\sqrt{25}$ **= 5** (koska 5 · 5 = 25).

$\sqrt{16}$ **= 4** (koska 4 · 4 = 16)

$\sqrt{100}$ **= 10** (koska 10 ·10 = 100).

TEHTÄVIÄ

Tehtävä 31. Itseisarvo

Koodaa ohjelma, joka tulostaa käyttäjän antaman luvun itseisarvon:

a. kirjoita ohjelman alkuun *from math import **
b. ohjelma pyytää käyttäjältä luvun
c. ohjelma tulostaa sekä annetun luvun ja sen itseisarvon

Tallenna ohjelma KOODIT-kansioon nimellä *itseisarvo* ja testaa ohjelmaa positiivisilla ja negatiivisilla luvuilla. Ohjelman tulosteet voisivat näyttää seuraavalta:

Syötä kokonaisluku: *-25*
Luvun *-25* itseisarvo on *25*

Tehtävä 32. Suurin yhteinen tekijä, gcd

Syt on hyödyllinen esimerkiksi murtolukujen supistamisessa. Murtoluvun osoittaja ja nimittäjä saadaan supistettua yksinkertaisimpaan muotoonsa.

Koodaa ohjelma, joka tulostaa kahden kokonaisluvun suurimman yhteisen tekijä (syt). Ohjelma kysyy kaksi kokonaislukua, laskee ja tulostaa lukujen suurimman yhteisen tekijän (gcd). Tallenna ohjelma KOODIT-kansioon nimellä *syt*.

Selvitä ohjelman avulla, mikä on alla olevien murtolukujen osoittajan ja nimittäjän suurin yhteinen tekijä. Supista alla esitetyt murtoluvut yksinkertaisimpaan muotoonsa.

```
# tulostaa kahden kokonaisluvun syt:n
from math import *
os = int(                    )
nim =  int(                  )
syt = gcd(    ,    )
print ("Lukujen",os,nim,"suurin yhteinen tekijä on ",syt)
```

$\frac{8}{16}$ syt(__) , supistettuna ____

$\frac{72}{90}$ syt (__), supistettuna ____

$\frac{36}{108}$ syt(__), supistettuna ____

Tehtävä 33. Murtoluvun supistaminen

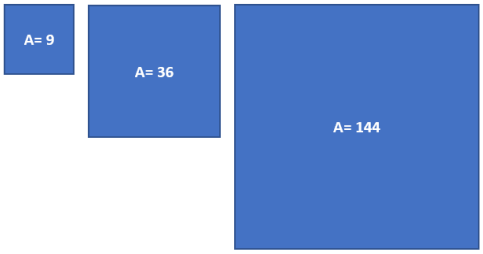
Jatka tehtävän 32 koodia siten, että ohjelma myös supistaa osoittajan (os) ja nimittäjän (nim) suurimmalla yhteisellä tekijällä. Ohjelma tulostaa lopuksi supistetun murtoluvun oheisen mallin mukaisesti.

```
Anna osoittaja: 36
Anna nimittäjä: 108
Lukujen 36 108 suurin yhteinen tekijä on 36
Murtoluku 36 / 108  on supistettuna 1 / 3
```

Pythonissa osamäärä saadaan / -operaattorilla, mutta tulos on desimaaliluku. Siksi jakolasku voidaan 'tyypittää' kokonaisluvuksi **int**-muunnoksella, esimerkiksi **int(5/2) = 2**.

Tehtävä 34. Neliön sivu ja pinta-ala, neliöllinen kasvu

Neliön sivun pituus voidaan laskea $\sqrt{A}$, kun neliön pinta-ala A tunnetaan. Esimerkiksi neliön pinta-ala on 64 cm^2. Tällöin neliön sivun pituus on $\sqrt{64}$ = 8 cm. Koodaa ohjelma, joka kysyy neliön pinta-alan, laskee neliön sivun pituuden (sqrt) ja tulostaa vastauksen. Tallenna ohjelma KOODIT-kansioon nimellä *sivunpituus*. Selvitä ohjelman avulla seuraavien neliöiden sivun pituudet.



Mitä huomaat sivujen pituuksien suhteesta? Entä, mitä havaitset pinta-alojen suhteesta?

_____

_____

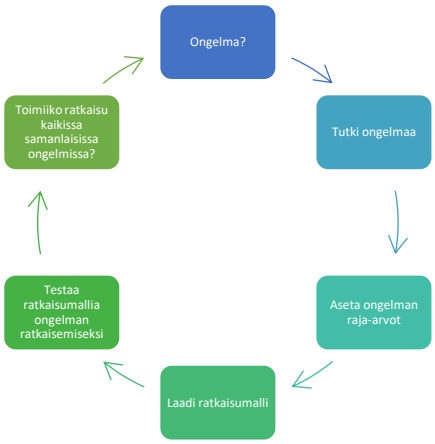
PERUSMATEMATIIKKAA OHJELMOIMALLA

matemaattisen ohjelmoinnin harjoituskirja 8-luokalle

Oppilaan nimi:_____

Python-ohjelmointia 8. vuosiluokan matematiikassa

Ohjelmoinnissa tarvitaan useita erilaisia rakenteita, joiden avulla algoritmeista saadaan yksinkertaisempia ja tehokkaampia. Ehtolauseilla ohjelma saadaan haarautumaan yhden tai useamman ehdon mukaan. Toistolauseilla jotakin ohjelman osaa toistetaan joko ennalta määrätty tai määräämätön määrä. Toistolauseilla vältetään pitkien peräkkäisten komentorivien kirjoittamiselta.



Jos haluat kerrata ja palauttaa mieleesi Python-ohjelmoinnin perusasioita 7-vuosiluokalta, niin voit palata aiempiin tehtäviin ja kerrata unohtuneet asiat. Myös 8-vuosiluokan ohjelmointitehtävissä on paljon matematiikassa opittuja asioita, joten ohjelmoinnilla voit vahvistaa samalla matematiikan taitojasi.

Tehtäväluettelo:

1. Merkkijono, alimerkkijono, merkki

Opit uutta:
- merkkijonon pituus
- alimerkkijono
- merkin poimiminen merkkijonosta
- merkkiliteraalit *\n, \t, \"*

Osaat jo:
- Python-ohjelmoinnin perusteita:

merkkijonot, lukutyypit, tyypinmuunnos, asetuslauseen, muuttujan arvon asetus, tulostaa merkkijonoja ja muuttujien arvoja, pyöristää desimaalilukuja, käyttää joitakin math-kirjaston toimintoja

Pythonissa merkkijonon sisältöä ei voida muuttaa luonnin jälkeen. Jos sisältöä halutaan muuttaa, tulee merkkijonon arvo asettaa uudelleen ja vanha arvo menetetään.
Merkkijonolla on **pituus** eli merkkien lukumäärä. Se saadaan selville ***len(jono)*** funktiolla, joka palauttaa merkkijonon *jono* pituuden kokonaislukuna.
Merkkijonon pituutta voidaan käyttää toistorajana esimerkiksi silmukoissa eli toistolauseissa.
Merkkijonon merkeillä on **paikkaindeksi**, joka kertoo merkin paikan merkkijonossa. Paikkaindeksointi alkaa 0:sta ja viimeisen merkin paikka on *pituus – 1*.

Merkkijonon paikkaindeksejä voi tarkastella myös takaperin, jolloin viimeisen alkion paikkaindeksi on -1 ja ensimmäisen merkin paikka on merkkijonon pituuden vastaluku.

```
pituus = len(jono)    # pituus on 10
```

Yksittäinen merkki voidaan poimia merkkijonosta indeksin avulla. Poimittavan merkin indeksi merkitään **hakasulkeisiin**. Esimerkiksi yllä esitetystä jonosta poimittaessa merkit ***jono[0], jono[1], jono[3]*** ja ***jono[9]*** saataisiin poimittua merkit *H a k a.* Lopusta alkuun poimittuna esimerkiksi *jono[-1], jono[-3], jono[-7]* ja *jono[-9]* saadaan merkit *a i k a*.

```
# poimitaan merkkijonosta
jono = "Halkaisija"
print (jono[0],jono[1],jono[3],jono[9])
print (jono[-1],jono[-3],jono[-7],jono[-9])
```

```
H a k a
a i k a
```

Alimerkkijono saadaan poimittua merkkijonosta merkitsemällä hakasulkeisiin ensimmäisen merkin ja viimeisen merkin indeksit kaksoispisteellä erottaen. Huomaa, että viimeisen indeksin merkki ei tule mukaan poimittavaan alimerkkijonoon. Esimerkiksi alimerkkijono ***jono[1:8]*** tarkoittaa merkkejä *a l k a i s i*.

```
# alimerkkijonoja
jono = "Halkaisija"
print (jono[1:8], jono[3:6])
```

```
alkaisi kai
```

Merkkijonoon voidaan lisätä ns. **merkkiliteraaleja** eli ohjausmerkkejä. Ne vaikuttavat merkkijonon tulostumiseen. Tässä yhteydessä tutustumme kolmeen merkkiliteraaliin ja niiden toimintaan:

- \n uusi rivi rivinvaihdolla
- \t vaakasisennys, tabulaattori
- \” lainausmerkki

```
# merkkiliteraaleja
print ("Rivin\nvaihto\n\tsisennetty\ttabulaattorilla")
print ("\"lainausmerkeissä tekstiä\"")
```

```
Rivin
vaihto
        sisennetty      tabulaattorilla
"lainausmerkeissä tekstiä"
```

TEHTÄVIÄ

Tehtävä 1. Merkkijonon merkkien paikka, indeksointi

Oheisessa koodissa on esitelty merkkijono *mjono*. Selvitä, mitkä sanat tulostuvat *mjono* merkkijonosta poimituista merkeistä. Kirjoita kirjaimet viivoille.

```
# poimitaan merkkejä

mjono = "PROSENTTIYKSIKKÖ"

print (mjono[1],mjono[4],mjono[8],mjono[5],mjono[2])  __ __ __ __ __

print (mjono[9],mjono[3],mjono[10],mjono[8],mjono[8]) __ __ __ __ __

print (mjono[-1],mjono[-4],mjono[-5],mjono[-8],mjono[-11]) __ __ __ __ __
```

Tarkista tulostuvat sanat koodaamalla ohjelma. Tallenna valmis ohjelma KOODIT-kansioon nimellä *poimittuja*.

Tehtävä 2. Merkkijonon pituus, tulostus takaperin

Koodaa ohjelma, joka pyytää käyttäjältä kaksi merkkijonoa; *eka* ja *toka*. Ohjelma tulostaa merkkijonojen pituudet. Lopuksi tulostetaan molemmat merkkijonot takaperin. **VIHJE:** merkkijono saadaan tulostettua takaperin ottamalla käyttöön kolmas indeksi, ns. 'harppaus'. Esimerkiksi *eka="TIMO",* niin ***print (eka[: :-1])*** tulostaa merkkijonon *eka* takaperin *OMIT.* Tallenna ohjelmasi KOODIT-kansioon nimellä *takaperin.* Ohjelman toiminta näyttää seuraavalta:

```
Anna merkkijono: iso rikas sika sökösakissa kirosi
Anna toinen merkkijono: ollapa pallo
Merkkijonojen pituudet ovat 33 ja 12
Takaperin:
isorik assikasökös akis sakir osi
ollap apallo
```

Tehtävä 3. Viipale merkkijonosta

Täydennä alla olevaan ohjelman indeksiviittaukset siten, että ohjelma tulostaa allekkain merkkijonon *lause* merkit seitsemän merkin mittaisissa alimerkkijonoissa eli viipaleissa. **VIHJE**: Kun alimerkkijonosta jättää toisen indeksin pois, tulostuu merkit aloituskohdasta merkkijonon loppuun. Esim. *ali[10:]* tarkoittaa merkkejä indeksistä 10 merkkijonon loppuun. Ole tarkkana indeksien kanssa, jotta kullekin riville tulostuu seitsemän merkkiä. Tallenna valmis ohjelma KOODIT-kansioon nimellä *viipaleet.*

```
# merkkijono viipaleina
lause = "MATEMATIIKKAA ON KAIKKIALLA!"

print (lause[ : ])
print (lause[ : ])
print (lause[ : ])
print (lause[ : ])
```

Tulostuksen pitää näyttää seuraavalta:

```
MATEMAT
IIKKAA
ON KAIK
KIALLA!
```

Huomaa, että myös välilyönnit ovat merkkejä.

Tehtävä 4. Merkkijonojen pituuksien suhde

Koodaa ohjelma, joka kysyy käyttäjältä kaksi merkkijonoa *mjono1* ja *mjono2*. Ohjelma tulostaa **prosenttilukuna**, kuinka monta prosenttia merkkijonon *mjono1* pituus on merkkijonon *mjono2* pituudesta.

Tallenna toimiva ohjelma KOODIT-kansioon nimellä *pituussuhde*. Ohjelman toiminta näyttää seuraavalta:

```
Kirjoita jokin merkkijono: keskuskulma
Kirjoita vielä toinen merkkijono: tangenttikulma
Ensimmäisen merkkijonon pituus on 78.6 %:a toisesta merkkijonosta
```

Muistathan, että desimaalilukuja voidaan pyöristää ***round(x,n)***-komennolla, jossa *x* on jokin float-tyyppinen luku ja *n* on esitettävien desimaalien määrä. Esimerkiksi *round*(2.4567,1*)* antaa tulokseksi 2.5.

Pyöristä merkkijonojen suhdeluku yhden desimaalin tarkkuuteen.

Prosenttiluku saadaan suhdeluvusta, eli jakolaskulla. Esimerkiksi luku 3 on luvusta 5 suhdelukuna $\frac{3}{5} = 0.6$, joka on prosenttilukuna **60%**.

2. Vertailuoperaattorit ja totuusarvot, loogiset operaatiot and ja or

Opit uutta:
- vertailuoperaattorit
- totuusarvot *True* ja *False*
- *and* ja *or* loogiset operaatiot

Osaat jo:
- tyypinmuunnokset lukutyypeiksi
- input-lauseen
- yhdistää merkkijonoja ja muuttujia tulostuslauseessa

Pythonissa tehdään suuruusvertailuja **vertailuoperaattoreilla**. Seuraavassa luettelossa esitetään vertailuoperaattorit sekä operaattorin selitys. Vertailun tulos on arvoltaan aina joko *True* tai *False* eli tosi tai epätosi. Lukuarvoina näitä vastaavat 1 (*True*) ja 0 (*False*).

- **<** pienempi kuin
- **<=** pienempi tai yhtä suuri kuin
- **>** suurempi kuin
- **>=** suurempi tai yhtä suuri kuin
- **==** yhtä suuri kuin, samat
- **!=** erisuuri kuin, eri
- **is** on sama kuin, tarkistaa myös vertailtavien arvojen tyypin
- **is not** on eri kuin, tarkistaa vertailtavien arvojen tyypin

Suurin osa vertailuoperaattoreiden käytöstä on loogisesti selvää, esimerkiksi vertailun **(1 < 2)** tulos on *True*. Poikkeuksellinen on **!=** eli erisuuri kuin -symboli, joka eroaa matematiikan vastaavasta symbolista ≠. Yhtä suuri kuin **==** vertailua ei saa sekoittaa asetuslauseen operaattoriin **=**.

```
# vertailujen totuusarvot
a = b = 1
c = 5
d = 5.0 # float-tyyppiä
print (a == b) # tulostaa True
print (b != c) # True
print (6+a >= b) # True
print (c is d) # False, koska c ja d erityyppisiä
```

```
True
True
True
False
```

Vertailulausekkeita voidaan ketjuttaa peräkkäin **Boolen algebran** operaatioilla. Boolen algebran loogiset operaatiot ovat **and**, **or** ja **not**. Loogisilla operaatioilla ketjutetun vertailulausekkeen arvo määräytyy Boolen algebran sääntöjen mukaisesti. Loogisilla operaatioilla voidaan vertailla vain totuusarvoisia (True/False) operandeja. Loogisen operaation tulos on myös totuusarvoinen. Alla taulukossa on esitetty totuusarvoisten (0/1) muuttujien a ja b loogisten operaatioiden arvot operaattoreilla and, or, not. Operaattori not kääntää arvon vastakkaiseksi, esimerkiksi not(1) = 0.

a	b	a or b	a and b	not a	not b
0	0	0	0	1	1
0	1	1	0	1	0
1	0	1	0	0	1
1	1	1	1	0	0

Pohdi lauseita:

*A. Jos on kylmä **ja** sataa, niin ota sateensuoja.*
*B. Jos on kylmä **tai** sataa, niin ota sateensuoja.*

Kummassa tapauksessa A vai B saatat ottaa sateensuojan mukaan turhaan?

Esimerkkinä loogisten operaatioiden ja vertailuoperaatioiden käytöstä esitetään seuraava ohjelma. Ohjelma tulostaa vertailulausekkeen ja loogisen lausekkeen yhdistelmän totuusarvoja.

```
# suuruusvertailuja ja loogisia operaatioita
x = float(input("Syötä luku x: "))
y = float(input("Syötä luku y: "))
print ("Lukuihin liittyviä väittämiä...")
print ("Molemmat positiivisia: ",x >= 0 and y >= 0)
print ("Luvut ovat eri suuria ja \nainakin toinen on pienempi kuin 10: ",x != y and x < 10 or y<10)
print ("Ainakin toinen luku on parillinen: ", x%2 == 0 or y%2 == 0)
```

```
Syötä luku x: -8
Syötä luku y: 9
Lukuihin liittyviä väittämiä...
Molemmat positiivisia:  False
Luvut ovat eri suuria ja
ainakin toinen on pienempi kuin 10:  True
Ainakin toinen luku on parillinen:  True
```

And-operaatiossa molempien vertailulausekkeiden on oltava tosia, jotta tulos on *True*. Or-operaatiossa ainakin toisen on oltava tosi, jotta tulos on *True*.

TEHTÄVIÄ

Tehtävä 5. Tutkitaan totuusarvoja, True tai False

Tutki alla olevaa koodia ja päättele, mitä totuusarvoja loogiset lausekkeet tulostavat kullekin riville. Kirjoita päättelysi tulos näkyviin viivoille.

Koodaa ohjelma ja tarkista tulokset. Tallenna ohjelma KOODIT-kansioon nimellä *totuusarvot*.

```
# tutki totuusarvot
x = -7
y = 2
print (y > x)                                  totuusarvo _____

print (x*y < 0 and abs(x*y)>0) # abs on itseisarvo totuusarvo _____

print (not(x==y) or (x != y))                  totuusarvo _____

print (x%2 == 0 and y%2 != 0)                  totuusarvo _____
```

Tehtävä 6. Vertailuoperaattorit < ja > sekä ==

Vertailuoperaattoreita voidaan käyttää myös **merkkijonojen vertailussa**. Esimerkiksi ”a” < ”b” vertailun tulos on *True*, koska kirjain a on aakkosissa ennen b kirjainta. Samoin voidaan vertailla esimerkiksi nimien aakkosjärjestystä. Vertailu ”Timo” < ”Tomi” on myös *True*.

Koodaa ohjelma, joka kysyy kaksi etunimeä ja tallentaa nimet muuttujiin *nimi1* ja *nimi2*. Ohjelmassa vertaillaan etunimien järjestys. Nimien aakkosjärjestysvertailun tulos ilmoitetaan totuusarvoilla *True* tai *False*.

Tallenna ohjelma KOODIT-kansioon nimellä *etunimet*. Ohjelman toiminta näyttää seuraavalta:

```
Anna nimi1: Aino
Anna nimi2: Aini
Aino on aakkosissa ennen nimeä Aini tulos: False
Aini on aakkosissa ennen nimeä Aino tulos: True
```

LISÄPULMA: Miten ohjelmassa saadaan tutkittua, onko käyttäjä syöttänyt saman etunimen kahteen kertaan?

(Voit palata takaisin 7-luokan tehtäviin. Tehtävässä 6 oli tehtävänä kirjoittaa algoritmi, joka järjestää etunimet aakkosjärjestykseen. Olisiko vertailuoperaattoreista apua tehtävän ratkaisemisessa?)

Tehtävä 7. Merkkijono lukutyypiksi, jakojäännösoperaattori %

Näppäimistöltä lukuja syötettäessä on tehtävä **tyypinmuunnos** lukutyypiksi. Jos esimerkiksi halutaan muuttujaan tallentaa kokonaisluku, tehdään tyypin muunnos *int*-tyypiksi seuraavasti:
luku = int(input(”Syötä luku: ”)).

Koodaa ohjelma, joka pyytää käyttäjää syöttämään **kokonaisluvun**. Ohjelmassa tutkitaan, onko luku parillinen (eli jaollinen luvulla 2). Parillisuustestin tulos ilmoitetaan arvoilla *True* tai *False.*

Tallenna ohjelma KOODIT-kansioon nimellä *onkoParillinen*. Ohjelman toiminta näyttää seuraavalta:

```
Syötä kokonaisluku: 123
VÄITE: luku 123 on parillinen, tulos:  False
```

LISÄPULMA: Onko kahden parittoman luvun **summa** aina parillinen?

_____

Tehtävä 8. Jaollisuustutkimus

Jakojäännösoperaattorilla **%** voidaan tutkia kokonaisluvun jaollisuutta. Esimerkiksi vertailulause 4%2 == 0 antaa tulokseksi *True*, koska osamäärän $\frac{4}{2}$ jakojäännös on 0 (*’jako menee tasan’*).

Koodaa ohjelma, joka pyytää käyttäjää syöttämään **kokonaisluvun**. Seuraavaksi ohjelma tulostaa totuusarvoiset tulokset luvun jaollisuudesta lukujen **2**, **3** ja **5** kanssa. Jaollisuus kunkin luvun kanssa testataan omalla rivillään. Tallenna ohjelma KOODIT-kansioon nimellä *jaollisuus*. Ohjelman toiminta on seuraavan kaltainen:

```
Syötä kokonaisluku: 90
Luku 90 on jaollinen luvulla 2, tulos: True
Luku 90 on jaollinen luvulla 3, tulos: True
Luku 90 on jaollinen luvulla 5, tulos: True
```

Mikä on pienin luku, joka on jaollinen luvuilla 2, 3 ja 5? _____

Millaisia lukuja luvut 2, 3 ja 5 ovat? _____

3. If, if-else, if-elif-else ehtolauseet

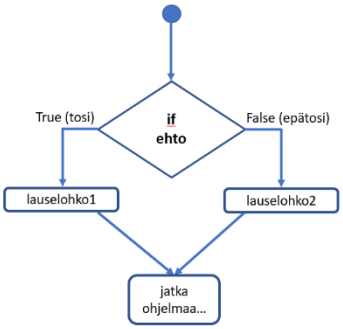
Opit uutta:
- if-lauseen
- if-else ja if-elif lauserakenteet
- ohjelmalohkon käsitteen

Osaat jo:
- vertailuoperaattorit
- totuusarvot *True* ja *False*
- *and* ja *or* loogiset operaatiot

Python-kielen ***if*** -valintarakenteella ohjelman suoritus saadaan haarautumaan jonkin ehdon mukaisesti. Ehdon totuusarvon ollessa *True* haarautuu ohjelma *if*-lohkoon. Ehdon totuusarvon ollessa *False* ohitetaan *if*-haara tai siirrytään mahdolliseen *else*-haaraan. *Else*-haara ei ole pakollinen, jolloin ohjelman suoritus jatkuu *if*-lauseen jälkeen seuraavasta lauseesta. *If*-lauseen syntaksi *else*-haara mukaan lukien on seuraava:

if totuustesti:
    lauselohko 1
else:
    lauselohko 2

if-lauseen totuustestin jälkeen tulee kaksoispiste. If-lauselohko kuuluu sisentää.



Esimerkiksi vertailun ”jos luku on pienempi kuin 0, tulosta ’negatiivinen’, muutoin tulosta ’positiivinen’”. Esimerkin mukainen algoritmi on esitetty ohessa Python-koodina.

If-ehtolause voidaan esittää vuokaaviona. If-ehto on esitetty salmiakkikuviona, josta ohjelman suoritus haarautuu ehdon totuusarvon mukaan. Epätosi tapauksessa suoritetaan niin sanottu *else* haara.

```
# if-else esimerkki
a = float(input("Syötä luku: "))
if a < 0:
   print ("Negatiivinen luku")
else:
   print ("Positiivinen luku")
```

```
Syötä luku: 2020
Positiivinen luku
```

if-lauseessa lauselohko sisentyy. Samoin else-haaran lauselohko sisentyy. **Tätä sisennystä ei saa poistaa**. Muuten tulkki antaa virheilmoituksen.

Else-haara ei ole pakollinen. Jos se jätetään pois, niin ohjelman suoritus jatkuu if-lohkon jälkeen seuraavasta rivistä.

```
# elif-else esimerkki
a = float(input("Syötä luku: "))
if a < 0:
   print ("Negatiivinen luku")
elif a == 0:
   print ("Luku on nolla")
elif a>0:
   print ("Luku on positiivinen")
else:
   print ("Tähän haaraan ei päästä")
```

```
Syötä luku: 0
Luku on nolla
```

Toisistaan riippuvat, **peräkkäiset totuustestaukset** saadaan lisäämällä if-lohkon perään yksi tai useampi *elif*-haara. *Elif* tulee sanoista else if. Rakenteesta suoritetaan se *elif*-haara, jossa ehto on tosi (*True*) ja kaikki muut *elif*-haarat ohitetaan. *If-elif*-rakenteen loppuun voidaan kirjoittaa *else*-haara, joka suoritetaan siinä tapauksessa, että mitään *elif*-haaroista ei suoritettu. Jokin valintarakenteen *if-*, *elif-* tai *else*-haaroista suoritetaan aina ja muut ohitetaan.

Ohjelmalohko on useissa ohjelmointikielissä syntaksin eli kieliopin kannalta tärkeä käsite. Se on yhteenkuuluvien lauseiden joukko, jotka suoritetaan omana kokonaisuutena muusta ohjelmakoodista erotettuna. Esimerkiksi Java-ohjelmointikielessä ohjelmalohko esitetään aaltosulkeiden välissä { }. Pythonissa käytetään lauselohkon sisennystä.

TEHTÄVIÄ

Tehtävä 9. Tulostus nimen pituuden ehdolla, if-lause

Koodaa ja täydennä alla esitetty ohjelma. Tehtäväsi on kirjoittaa print-lauseiden korostettuihin kohtiin if-lauseessa suoritetun vertailun tulos: *”nimessä on A- tai a-kirjain”, ”Nimessä on alle 4 kirjainta”, ”nimessä ei ole a-kirjainta”* jne.

```
# if-elif-tehtävä
etunimi = input("Kirjoita etunimesi: ")

if len(etunimi) < 4:
    print ("                    ")

elif len(etunimi)>=4 and len(etunimi)<=6:
    print ("                    ")

elif len(etunimi)>6:
    print ("                    ")

if "a" in etunimi or "A"in etunimi:
    print ("                    ")
else:
    print ("                    ")
```

Tallenna ohjelma KOODIT-kansioon nimellä *nimitutkimus.*

Tehtävä 10. Yhteenlaskutesti

Koodaa ohjelma, joka pyytää käyttäjää syöttämään kaksi desimaalilukua (float) ja tallentaa ne muuttujiin *a* ja *b*. Ohjelma kysyy käyttäjältä lukujen summan ja tarkistaa vastauksen oikeellisuuden if-lausetestillä. Oikeasta vastauksesta saa ”Oikein”, väärästä vastauksesta viestin ”Ei mennyt nappiin”.

Tallenna ohjelma KOODIT-kansioon nimellä *lukusummat*. Ohjelman toiminta on seuraava:

```
Anna luku a: 2.5
Anna luku b: 3.5
Laske 2.5 + 3.5
Vastaus: 6
Oikein!
```

tai

```
Anna luku a: 3.2
Anna luku b: 3.3
Laske 3.2 + 3.3
Vastaus: 6
Ei mennyt nappiin
```

Tehtävä 11. Itseisarvo, abs

Koodaa ohjelma, joka

a. kysyy käyttäjältä kaksi **kokonaislukua** (int) ja tallentaa ne muuttujiin *x* ja *y*
b. vertailee lukujen itseisarvoja **|x|** ja **|y|** (Pythonissa esim. *abs(-5) on 5*)
c. tulostaa itseisarvoltaan suuremman luvun arvon
d. tulostaa myös tiedon siitä, jos lukujen itseisarvot ovat yhtä suuret
e. tallenna ohjelma KOODIT-kansioon nimellä *itseisarvovertailu*.
f. testaa ohjelman toiminta esimerkiksi luvuilla
   ii. -25 ja 20
   iii. -33 ja -45
   iv. -90 ja 90

Ohjelman toiminta näyttää seuraavalta:

```
Anna luku x: -10
Anna luku y: 20
Luku 20 on itseisarvoltaan suurempi
```

tai

```
Anna luku x: -100
Anna luku y: 100
Luvut -100 100 ovat itseisarvoltaan yhtä suuret
```

Tehtävä 12. Simon lukuvertailuissa korjattavaa

Simo on koodannut huolimattomasti alla olevan ohjelman. Ohjelmassa on syntaksivirheitä ja ajatusvirheitä. **Korjaa virheet** ja koodaa ohjelma toimivaksi. Tallenna se KOODIT-kansioon nimellä *Simon_lukuvertailu*.

```
# lukuvertailu ja suuruusero

x = input(int(Syötä luku x)
y = input(int(Syötä luku y)
if y = x
    print (y,'on pienempi tai yhtä suuri kuin',x)

else if:
    print (y,'on suurempi kuin',x)

print ("Lasketaan lukujen suuruusero itseisarvon avulla")

etaisyys == |x-y|

print (Lukujen suuruusero on: etaisyys)
```

4. Satunnaislukuja ja muita random-kirjaston toimintoja

Opit uutta:
- random-satunnaislukukirjaston käytön
- satunnaisuuden käytön matemaattisissa ohjelmissa
- *random, randint, randrange, choice* funktiot

Osaat jo:
- if-, elif- ja else lauserakenteet
- vertailuoperaattorit
- ohjelmalohkon käsitteen

Pythonissa on *math*-kirjaston lisäksi useita muitakin valmiita ohjelmakirjastoja. Esimerkiksi ***random***-aliohjelmakirjastolla saadaan satunnaislukujen lisäksi muitakin satunnaistoimintoja. Satunnaisuutta voi hyödyntää esimerkiksi peleissä ja opetusohjelmissa. Kaikki random-kirjaston käskyt saadaan käyttöön kirjoittamalla ohjelmakoodin alkuun lause ***from random import ****.

Alla on lueteltu muutamia random-kirjaston käskyjä selityksineen. Ohessa on koodiesimerkki *random*-kirjaston käskyjen käytöstä ja niiden tuottamasta satunnaisluvuista. Huomaa, että ohjelmassa on käytetty listaa. **Lista** käsitellään tarkemmin myöhemmin, mutta tässä yhteydessä se on vain esimerkkinä *sample*-funktion käytöstä.

Random-kirjaston funktio	**Toiminto**
randrange(*loppu*)	Tuottaa satunnaisen kokonaisluvun väliltä 0 – loppu
randrange(*alku, loppu*)	Tuottaa satunnaisluvun väliltä alku – loppu
randint(*alku,loppu*)	Tuottaa satunnaisen kokonaisluvun väliltä alku – loppu
sample(*lista, lkm*)	Tuottaa *lkm* satunnaista alkiota listasta *lista*
random()	Tuottaa satunnaisen float-luvun väliltä 0.0–1.0

```
# random-kirjaston käyttöesimerkkejä
from random import *
nimet = ['Eetu','Jutta','Pilvi','Maija','Rasmus','Emilia','Ella','Ville']
print (randrange(10)) # kokonaisluku väliltä 0-10
print (randint(1,100)) # satunnaisluku väliltä 1-100
print (sample(nimet,2)) # kaksi satunnaista alkiota nimet-listasta
print (random()) #satunnaisluku 0.0 - 1.0
```

```
3
98
['Jutta', 'Ella']
0.12559547441486318
```

Ohjelmoinnissa **funktioksi** kutsutaan aliohjelmaa, joka palauttaa suorituksen tuloksena jonkin arvon. Random-kirjaston ”käskyt” ovat funktioita. Ne palauttavat mm. satunnaislukuja.

Alla on esitetty yhteenlaskuohjelman alkuosa. Ohjelma kysyy käyttäjältä ylä- ja alarajan lukualueelle, josta yhteenlaskettavat satunnaisesti (randint) ’arvotaan’. Ohjelmasta puuttuu käyttäjän antaman vastauksen tarkistus. Tarkistuksessa tarvitaan if-else -rakennetta. Tämä jää harjoitustehtäväksi.

```
# yhteenlaskuja päässälaskuna
from random import *
print ("Lasketaan kahden satunnaisen kokonaisluvun summia")
alku = int(input("Anna kokonaislukuna lukualueen alaraja: "))
loppu = int(input("Anna kokonaislukuna lukualueen yläraja: "))
a, b = randint(alku,loppu), randint(alku,loppu)
print ("Laske",a,"+",b)
vastaus = int(input("Vastaus: "))
# tähän pitäisi koodata oikeellisuustarkistus

# ja palaute oikeasta tai väärästä vastauksesta
```

Rivillä *a,b = randint(alku,loppu), randint(alku,loppu)* asetetaan samanaikaisesti satunnaislukuarvot muuttujille *a* ja *b*.

TEHTÄVIÄ

Tehtävä 13. Arvaa luku, randint

Koodaa ohjelma, joka arpoo satunnaisluvun (int) väliltä 1-10. Seuraavaksi ohjelma pyytää käyttäjää arvaaman luvun. Ohjelma vertaa käyttäjän antamaa lukua arvottuun lukuun ja kertoo, oliko arvaus oikein, oliko arvaus liian pieni luku tai oliko arvaus liian suuri luku. Satunnaisluvun tutkimiseen tarvitaan if- elif -else -rakennetta.

Tallenna ohjelma KOODIT-kansioon nimellä *lukuarvaus*. Testaa ohjelmaa useita kertoja. Millä todennäköisyydellä arvauksesi osuu oikeaan?

Ohjelman toiminta on seuraava:

```
Arvaa luku 1-10: 3
Arvasit liian pienen luvun
Oikea luku oli 5
```

tai

```
Arvaa luku 1-10: 7
Arvasit liian suuren luvun.
Oikea luku oli 1
```

tai

```
Arvaa luku 1-10: 2
Oikein meni!
```

Tehtävä 14. Yhteenlaskutehtävän tarkistus

Edellisellä teoriasivulla on esitetty **yhteenlaskuohjelman** alkuosa. Koodaa ja täydennä ohjelmaan oikeellisuustarkistus. Oikeellisuustarkistuksessa verrataan lukujen summaa käyttäjän antamaan vaihtoehtoon. Tarkistuksen tuloksen mukaan tulostetaan viesti vastauksen oikeellisuudesta. (Vrt. tehtävä 5.)

Tallenna täydennetty ohjelma KOODIT-kansioon nimellä *yhteenlaskut*. Testaa ohjelman toiminta useita kertoja.

```
# yhteenlaskuja päässälaskuna
from random import *
print ("Lasketaan kahden satunnaisluvun (kokonaislukuja) summia")
alku = int(input("Anna kokonaislukuna lukualueen alaraja: "))
loppu = int(input("Anna kokonaislukuna lukualueen yläraja: "))
a, b = randint(alku,loppu), randint(alku,loppu)
print ("Laske",a,"+",b)
vastaus = int(input("Vastaus: "))
# tähän pitäisi koodata oikeellisuustarkistus

# ja palaute oikeasta tai väärästä vastauksesta
```

Tehtävä 15. Kaksi arpanoppaa

Koodaa ohjelma, joka simuloi kahden nopan (silmäluvut 1-6) heittoa. Ohjelma kysyy käyttäjältä, että heitetäänkö yksi noppa kerrallaan vai molemmat yhtä aikaa. Valinnat voivat olla esimerkiksi 1 = yksi kerrallaan, 2 = molemmat yhtä aikaa. Ohjelma tulostaa noppien silmäluvut. Ohjelmassa tarvitaan randint-funktion lisäksi if-elif-else -rakennetta.

Tallenna ohjelma KOODIT-kansioon nimellä *arpanopat*. Testaa ohjelma toiminta, joka näyttää seuraavalta:

```
Heitetään kahta noppaa
Yksi kerrallaan = 1, molemmat yhtä aikaa = 2: 1
1. nopan silmäluku:  3
Paina enter...
2. nopan silmäluku:  2
```

tai

```
Heitetään kahta noppaa
Yksi kerrallaan = 1, molemmat yhtä aikaa = 2: 2
Noppien silmäluvut:  5 ja 6
```

tai

```
Heitetään kahta noppaa
Yksi kerrallaan = 1, molemmat yhtä aikaa = 2: 3
Valinta ei mahdollinen
```

Tehtävä 16. Simon kolikonheitto

Simo yrittää koodata kolikonheittopeliä, jossa on kaksi todennäköistä vaihtoehtoa; 1=*kruuna* tai 2=*klaava*. Simon koodissa kaikki ei ole ihan syntaksin mukaisesti. Tehtäväsi on korjata Simon koodi toimivaksi. Tallenna korjattu toimiva ohjelma KOODIT-kansioon nimellä *Simon_kolikonheitto.*

```
# kolikonheitto

random import kolikko

valinta = int(output("Arvaa Kruuna=1 tai Klaava=2: "))

kolikko = random(1,2)

if valinta = kolikko

print ("Hienosti arvattu! Nappiin meni!")

elif valinta ≠ kolikko

print ("Hupsis, ei osunut oikeaan")
```

5. While-toistolause, ikuinen silmukka

Opit uutta:
- while-toistolauseen eli silmukan
- ikuisen while-silmukan
- silmukan päättämisen *break*-komennolla

Osaat jo:
- if-ehtolauserakenteet
- satunnaisuuden käytön matemaattisissa ohjelmissa
- *random, randint, randrange, choice* funktiot

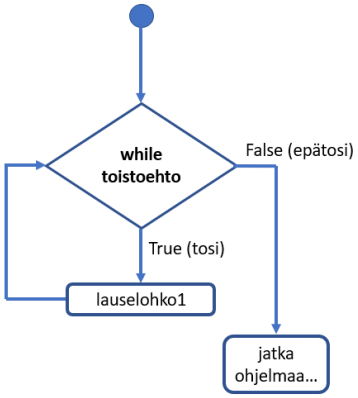
Silmukoilla eli toistolauseilla ohjelman jotain kohtaa voidaan toistaa useita kertoja. **While**-silmukassa on jokin totuusarvoinen toistoehto, jonka mukaan silmukkaa toistetaan tai siitä poistutaan.

***While*-lause** suorittaa toistolohkon lauseita niin kauan kuin lopetusehto on tosi eli *True*. Toistokertojen määrää ei vättämättä tiedetä ennalta. *While*-lauseen syntaksi on seuraava:

while toistoehto:

lauselohko1

While-lauseen määrittelyrivi päättyy kaksoispisteeseen. Toistettava lauselohko sisennetään.



Varatun sanan *while* jälkeen on toistoehto, joka päättyy kaksoispisteeseen. Toistettavat lauseet sisennetään lohkoksi. Seuraavassa ohjelmaesimerkissä syötetään parillisia lukuja. Jos käyttäjä syöttää parittoman luvun, päättyy silmukka lopetusehdon mukaisesti.

```
# syötetään parittomia lukuja.
luku = 2 # luvun alkuarvo, jotta silmukkaan päästään
while (luku%2 == 0):
   luku = float(input("Syötä parillinen luku: "))
   print ("...jatketaan...")
print ("Syöttämäsi luku on pariton.")
```

```
Syötä parillinen luku: 56
...jatketaan...
Syötä parillinen luku: 41
...jatketaan...
Syöttämäsi luku on pariton.
```

Silmukan toistoehto luku%2 == 0 tarkistaa, onko luku jaollinen 2:lla. Jos on, niin jakojäännös on silloin = 0.

While-lohko sisennetään samalla tavalla kuin if-lohkokin. Idle-editori tekee sisennyksen automaattisesti.

```
# ikuinen while-silmukka
summa = 0 # alustetaan summan arvo
while True:
   luku = input("Syötä luku (x lopettaa): ")
   if luku == 'x':
      break
   else:
      summa = summa + float(luku)
print ("Syöttämiesi lukujen summa on",summa)
```

```
Syötä luku (x lopettaa): 23
Syötä luku (x lopettaa): 1.5
Syötä luku (x lopettaa): x
Syöttämiesi lukujen summa on 24.5
```

Toisinaan ohjelma voi puutteellisen lopetusehdon vuoksi jäädä ikuiseen silmukkaan eli **luuppiin**. Luuppiin jäänyt ohjelma saadaan keskeytettyä **Ctrl+C** näppäinyhdistelmällä.

Ikuista silmukkaa käytetään myös tarkoituksella esimerkiksi, kun ohjelman käytön halutaan jatkuvan, kunnes käyttäjä haluaa sen itse lopettaa. Ikuinen silmukka saadaan kirjoittamalla *while*-lauseen lopetusehdoksi totuusarvo *True*. Ikuinen silmukka voidaan lopettaa if-lohkossa olevalla ***break*** -komennolla. Seuraava esimerkkiohjelma ikuisesta silmukasta lukee lukuja, kunnes käyttäjä syöttää lopetusmerkin 'x'. *Else*-haarassa lasketaan syötettyjen lukujen summa.

Syöte pitää muuttaa lukutyypiksi, jotta sitä voidaan käyttää laskutoimituksissa.

TEHTÄVIÄ

Tehtävä 17. Merkkijono while-silmukassa

Tässä tehtävässä tulostetaan merkkijono while-silmukassa. Merkkijonon pituus saadaan ***len()*** -funktiolla, esim. *len(sana).* While-toistossa on huolehdittava kuitenkin siitä, että **lopetusehto joskus saavutetaan**. Jos lopetusehtoa ei saavuteta, on tuloksena ikuinen silmukka. Silmukan lopettamiseen käytetään niin sanottua **silmukkalaskuria,** jonka arvoa kasvatetaan (tai vähennetään) jokaisella toistokerralla.

Koodaa alla esitetty ohjelma, tutki sen toiminta syöttämällä erilaisia sanoja. Tallenna ohjelma KOODIT-kansioon nimellä *sanasilmukka*.

```
# sanasilmukka
sana = input("Kirjoita jokin sana: ")
laskuri = 0  # silmukkalaskurin alustaminen
while (laskuri < len(sana)):
    print (sana[laskuri])
    laskuri += 1  # laskurin kasvatus
```

sana[laskuri] poimii muuttujan laskuri osoittaman merkin merkkijonosta. Esim. sana[0] viittaa ensimmäiseen kirjaimeen.

Tehtävä 18. Parilliset luvut ylärajalla

Koodaa ohjelma, joka tulostaa allekkain parillisia lukuja käyttäjän antamaan ylärajaan *raja* saakka. Muista laittaa ohjelman while-silmukkaan laskuri, ettei silmukasta tule ikuinen. Parilliset luvut tulostetaan if-lauseen totuusehdon mukaan ***laskuri%2 == 0***. Myös yläraja tulostetaan, jos se on parillinen.

Tallenna ohjelma KOODIT-kansioon nimellä *parillissilmukka*. Ohjelman toiminta näyttää seuraavalta:

```
Tulostetaan parillisia lukuja ylärajaan saakka
Anna yläraja: 10
0
2
4
6
8
10
```

Tehtävä 19. Lukuarvaus silmukassa

Tehtävässä 13 koodasit ohjelman, joka arpoo satunnaisluvun (int) väliltä 1-10 ja käyttäjän tulee arvata luku. Muuta ohjelmaa siten, että arvattavaa lukua pyydetään **ikuisessa silmukassa** (*while True:*) niin kauan, kunnes käyttäjä arvaa oikean luvun. Kun oikea luku arvataan, niin silmukka lopetetaan *break*-lauseella.

Tallenna ohjelma KOODIT-kansioon nimellä *silmukkalukuarvaus*. Testaa ohjelmaa useita kertoja. Kuinka monta arvausta tarvitset oikean luvun arvaamiseksi? Ohjelman toiminta on seuraava:

```
Arvaa luku 1-10: 5
Arvasit liian pienen luvun
Arvaa luku 1-10: 7
Arvasit liian pienen luvun
Arvaa luku 1-10: 9
Arvasit oikein!
```

Hakualgoritmia, jossa haetaan järjestetyn listan puolesta välistä, kutsutaan **binaari- eli puolitushauksi**. Tällä algoritmilla lukualue aina puolittuu ja haku nopeutuu. Vähiten arvauksia tarvitset, kun valitset luvun aina jäljelle jäävän lukujonon puolesta välistä.

Tehtävä 20. Kertolaskujen opetusohjelma

Koodaa kahden luvun kertolaskujen opetusohjelma. Ohjelman keskeiset toiminnot ovat seuraavat:

- **ikuisessa while**-silmukassa arpoo kaksi satunnaista kokonaislukua väliltä 1-10 ja tallentaa luvut muuttujiin m ja n. Muista aloittaa ohjelma rivillä *from random import **
- kysyy käyttäjältä lukujen m ja n tulon (m*n)
- testaa *vastauksen* oikeellisuuden if-lauseessa
    - jos *vastaus* on oikein, tulostetaan ”Oikein”
    - jos vastaus on väärin, tulostetaan ”Oikea vastaus on” sekä oikea vastaus
- kysyy, haluaako käyttäjä lopettaa ohjelman ”Haluatko lopettaa k/e?”
    - jos *lopetus* vastaus on ”k”, niin ohjelma lopetetaan break -lauseella.

Tallenna ohjelma KOODIT-kansioon nimellä *opetusohjelma*. Testaa ohjelmaa. Ohjelman toiminta on seuraava:

```
Laske 6 * 5 =
vastaus: 30
Oikein!
Haluatko lopettaa k/e? e
Laske 4 * 3 =
vastaus: 10
Oikea vastaus on 12
Haluatko lopettaa k/e? k
```

6. For in-toistolause, range-lukujono

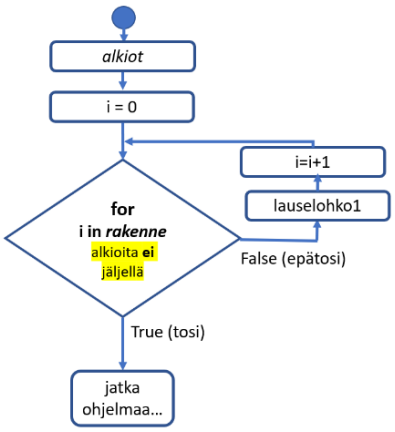
Opit uutta:
- *for in* -toistolauseen
- merkkijonon läpikäynti *for*-silmukassa
- *range* lukujonon

Osaat jo:
- while-toistolauseen eli silmukan
- ikuisen while-silmukan
- silmukan päättämisen *break*-komennolla

Toinen hyödyllinen silmukka on ***for*-silmukka**. *For*-silmukka eroaa *while*-silmukasta siinä, että *for*-silmukassa on etukäteen oltava tiedossa toistokertojen määrä. Esimerkiksi merkkijono tai lista voidaan askeltaa alkio kerrallaan *for*-silmukassa. Toistojen lukumäärä saadaan myös ***range()*** -funktiolla. (Funktio on Pythonissa valmiiksi ohjelmoit toiminto). For-lauseen syntaksi on seuraava:

for i in rakenne:

lauselohko1



Seuraavassa esimerkissä merkkijono tulostetaan merkki kerrallaan for-silmukassa. Muuttuja *merkki* 'juoksee' merkkijonon läpi merkki kerrallaan.

```
# for in -silmukka
mjono = "Omena"
for merkki in mjono:
    print (merkki)
```

```
O
m
e
n
a
```

Merkkijono tulostuu for-silmukassa merkki kerrallaan allekkain.

For-silmukan toistokertojen lukumäärä voidaan määrätä ***range()*** -funktiolla. *Range* luo automaattisesti lukujonon. *Range()*-funktiossa annetaan lukujonon alku- ja loppuarvot parametreina. Jos lukujonon arvojen halutaan 'hyppäävän', niin tarvitaan kolmaskin parametri, joka kertoo hypyn pituuden. *Range*-funktion syntaksi on seuraava:

range(alku, loppu,hyppy)

Aloitusarvo voidaan jättää pois, jolloin se on oletuksena 0. Parametri *loppu* ei sisälly generoitavaan lukujonoon. Esimerkiksi *range(6)* generoi lukujonon 0,1,2,3,4,5 ja *range(1, 100)* lukujonon 1–99. *Range()*-funktiossa voi olla kolmantena parametrina *hyppy*, jolla saadaan poimittua lukuja tietyin välein. Jos hyppy on negatiivinen, *range()* generoi laskevan lukujonon. Esimerkiksi *range(0, –20, –2)* generoi laskevan lukujonon *0–(–20)* kahden välein.

```
# for in -lukujono rangella
for i in range(5):
    print (i)
for luku in range(10,20,2):
    print (luku)
```

```
0
1
2
3
4
```

```
10
12
14
16
18
```

range(5) tulostaa lukujonon 0-4 ja *range(10,20,2)* tulostaa parilliset luvut 10-18. Huomaa, että loppuparametri ei tule mukaan lukujonoon.

Tehtävä 21. Merkkijono for-silmukassa

Koodaa ohjelma, joka kysyy käyttäjältä merkkijonon ja tulostaa sen merkki kerrallaan *for*-silmukassa. Tallenna ohjelma KOODIT-kansioon nimellä *merkkisilmukka*. Ohjelman toiminta näyttää seuraavalta:

```
Kirjoita jokin merkkijono: Einstein
E
i
n
s
t
e
i
n
```

Tehtävä 22. Lasketaan a-kirjaimet

Algoritmiin koodataan usein eri rakenteita; silmukoita (while, for) ja ehtolauseita (if). Koodaa ohjelma, jossa for-silmukassa käydään läpi käyttäjän syöttämä merkkijono. Silmukassa testataan, onko merkkijonossa ”a” kirjaimia. Silmukassa lasketaan a-kirjainten esiintymien lukumäärä. Ohjelmassa tarvitset mm.

- muuttujia *mjono, lukumaara*
- for-silmukka
- if-lause, jossa testataan *merkki == ”a”*
- jos ”a”-kirjain löytyy, kasvatetaan laskurin *lukumaara* arvoa.

Tallenna ohjelma KOODIT-kansioon nimellä *merkkilaskuri*. Testaa ohjelman toiminta. Ohjelman toiminta on seuraava:

```
Kirjoita merkkijono: Aavasaksa
Lasketaan a-kirjainten esiintymät
a-kirjaimia löytyi 4 kappaletta
```

Huomaa, että ”a”- ja ”A”-kirjaimet ovat ohjelmoinnissa eri kirjaimia. Mitä pitäisi lisätä if-lauseen vertailuehtoon, jotta myös ”A”-kirjaimet laskettaisiin mukaan?

Tehtävä 23. Säännöllisiä lukujonoja

Koodaa alla oleva ohjelma, joka tulostaa allekkain parilliset luvut 0:sta 40:een saakka laskusäännön *2*i* mukaan. Tallenna ohjelma KOODIT-kansioon nimellä *silmukkalukujonot*.

```
for i in range(21):
    print (2*i)
```

a. muuta laskusääntöä siten, että silmukassa tulostetaan parittomat luvut 1, 3, 5, 7, 9...,41.

b. muuta laskusääntöä siten, että lukujono tulostaa säännön *i**2* mukaan 40 lukua. Mikä on lukujonon viimeinen luku? _____

c. mikä laskusääntö saa tulostumaan luvut -1, 0, 3, 8, 15, 24, 35, ..., 399?

_____

d. mikä laskusääntö saa tulostumaan luvut 0, 0, 2, 6, 12, 20, 30, ..., 380

_____

Tehtävä 24. Merkkikuvioita

a. For-silmukassa voidaan myös merkkikuvioita. Koodaa oheinen ohjelma, joka tulostaa **silmukassa** viisikerroksisen #-merkkiportaikon. Tallenna ohjelma KOODIT-kansioon nimellä *forkuviot*.

```
for i in range(1,6):
    print (i*"#")
```

```
#
##
###
####
#####
```

b. muuta silmukan tulostussääntöä siten, että portaikko tulostuu toisin päin.

```
#####
####
###
##
#
```

c. muuta silmukan tulostussääntö siten, että se tulostaa alla olevan kuvan mukaisen kuvion. Kaksi eri merkkiä saadaan yhdistettyä (katenoitua) samalle riville **+** merkillä.

```
o#####
oo####
ooo###
oooo##
ooooo#
```

7. Murtolukuja fractions-kirjastolla

Opit uutta:

- murtolukujen esittämisen Pythonissa fractions-kirjastolla
- murtolukujen laskutoimituksia ohjelmoimalla
- *and* ja *or* loogiset operaatiot

Osaat jo:

- käyttää math-kirjaston funktioita
- while- ja for-silmukat sekä range lukujonon
- säännönmukaisuuksia, esim. lukujonoja ja merkkikuvioita

Murtolukujen esitys ja laskutoimitukset onnistuvat Pythonissa **fractions**-kirjaston avulla (fraction suom. murtoluku). Fractions -kirjasto otetaan käyttöön kirjoittamalla ohjelmakoodin alkuun ***from fractions import ****. Esimerkiksi murtoluku ½ saadaan lauseella Fraction(1,2). (Huomaa iso alkukirjain.)

Murtoluvut supistetaan automaattisesti, esimerkiksi Fraction(9,12) supistetaan muotoon $^3/_4$.

Murtolukujen laskutoimituksia voidaan suorittaa aritmeettisten operaattoreiden avulla. **Fractions-kirjastolla ei voida esittää sekalukuja**. Esimerkiksi sekaluku $2\frac{1}{5}$ tulee Pythonissa muuttaa murtomerkinnäksi Fraction(11,5).

```
# fraction-esimerkki
from fractions import *
s = Fraction(2,3) # 2/3
r = Fraction(3,4) # 3/4
t = Fraction(17,12) # 17/12
print (s,r,t)
print (s+r-t) # 17/12 - 17/12
print (s*r)   # 6/12 = 1/2
```

```
2/3 3/4 17/12
0
1/2
```

Murtoluvun voi asettaa muuttujan arvoksi asetuslauseella ja murtolukujen laskutoimitukset voidaan suorittaa kuten kokonais- ja desimaalilukumuuttujillakin. Esimerkiksi lauseet

```
from fractions import Fraction
a = Fraction(3,4)
b = Fraction(6/12)
```

luo murtoluvut $\frac{3}{4}$ ja $\frac{6}{12}$, joista jälkimmäisen Python-tulkki supistaa muotoon ½.

Seuraavassa ohjelmaesimerkissä käyttäjältä kysytään kahden murtoluvun **osoittaja** ja **nimittäjä**. Murtolukujen arvot asetetaan samalla rivillä. Lopuksi tulostetaan murtolukujen laskutoimitusten tuloksia supistettuna**.** Ohjelmassa on paljon aiemmin opittua ohjelmoinnin teoriaa. Osaatko selittää ohjelmarivien toiminnan rivi riviltä?

```
# murtolukulaskuja fractions-kirjastolla
from fractions import * # otetaan käyttöön murtolukukirjasto
os1 = int(input("osoittaja 1: ")) # ensimmäisen murtoluvun osoittaja
nim1 = int(input("nimittäjä 1: ")) # ensimmäisen murtoluvun nimittäjä
os2 = int(input("osoittaja 2: ")) # toisen murtoluvun osoittaja
nim2 = int(input("nimittäjä 2: ")) # toisen murtoluvun nimittäjä
# luodaan murtoluvut
mx , my = Fraction(os1,nim1), Fraction(os2,nim2) # luodaan murtoluvut mx ja my samanaikaisesti
print ("Syöttämäsi murtoluvut ovat",mx,my)
print ("Summa =",mx+my,",erotus =",mx-my)
print ("Tulo =",mx*my,",osamäärä =",mx/my)
```

TEHTÄVIÄ

Tehtävä 25. Murtolukujen yhteen- ja vähennyslaskuja Pythonilla

Ohessa on ohjelmakoodi, joka laskee murtolukujen *a* ja *b* yhteen- ja vähennyslaskut ja tulostaa vastaukset. Koodaa ohjelma ja tallenna se KOODIT-kansioon nimellä *murtolukulaskut.*

```
# Murtolukulaskuja
from fractions import Fraction
a = Fraction(3,4)
b = Fraction(6,12)
print ("Lasketaan murtolukujen",a,"ja",b,"laskutoimituksia")
print ("Summa", a+b)
print ("Erotus", a-b)
```

Laske seuraavat murtolukulaskut ja tarkista vastaukset ohjelman avulla. **Muuta sekamerkinnät epämurtoluvuksi**, jotta voit syöttää osoittajan ja nimittäjän Fraction-muuttujaan. Esimerkiksi $2\frac{1}{2}$ on epämurtolukuna $\frac{5}{2}$

$\frac{5}{6}+\frac{2}{3}=$ $\qquad$ $2\frac{1}{2}-1\frac{3}{4}=$

$\frac{7}{8}+\frac{2}{3}=$ $\qquad$ $10\frac{1}{2}-20\frac{3}{4}=$

Tehtävä 26. Murtolukujen kerto- ja jakolaskuja

Lisää tehtävääs 25 koodattuun murtolukulaskuohjelmaan myös lukujen a ja b **kertolaskulauseke** sekä **jakolaskulauseke**. Lisää koodiin myös näille **tulostuslauseet**. Laske alla esitetyt murtolukujen laskutoimitukset ja tarkista tulokset ohjelmaa käyttäen.

$\frac{5}{6}\cdot\frac{6}{5}=$ $\qquad$ $2\frac{1}{5}\cdot\frac{3}{4}=$

$\frac{6}{9}:\frac{2}{3}=$ $\qquad$ $5\frac{1}{2}:2\frac{3}{4}=$

Tehtävä 27. Kahden murtoluvun välissä oleva luku

Kahden murtoluvun välissä oleva murtoluku saadaan laskemalla osoittajat yhteen ja nimittäjät yhteen. Saatu vastaus supistetaan, jos mahdollista. Esimerkiksi lukujen $\frac{1}{3}$ ja $\frac{2}{3}$ välissä oleva luku on $\frac{1+2}{3+3}=\frac{3}{6}=\frac{1}{2}$.

Pythonissa **osoittaja** saadaan ***numerator***-komennolla ja **nimittäjä *denominator*** komennolla.

Koodaa ohjelma, joka selvittää kahden murtoluvun välissä olevan murtoluvun yllä kuvatulla laskutavalla. Murtoluvut asetaan muuttujien *a* ja *b* arvoiksi. Ohjelma laskee murtolukujen osoittajat yhteen ja nimittäjät yhteen. Lopuksi ohjelmassa tulostetaan annettujen murtolukujen välissä oleva murtoluku.

```
# murtoluvun osoittaja ja nimittäjä
from fractions import Fraction
a = Fraction(3,4)
os = a.numerator   # osoittaja
nim = a.denominator # nimittäjä
print ("Osoittaja",os,", nimittäjä",nim)
```

Muuta murtolukumuuttujien *a* ja *b* arvoja ja selvitä, mikä murtoluku on seuraavien murtolukujen välissä?

$\frac{1}{3}<-<\frac{1}{2}$ $\qquad$ $\frac{3}{5}<-<\frac{3}{4}$ $\qquad$ $\frac{5}{9}<-<\frac{2}{3}$

Huomaa, että mallikoodista puuttuvat kokonaan toinen murtoluku (b), osoittajien ja nimittäjien yhteenlasku sekä tulostukset.

Tehtävä 28. Juomatehtailijan pullolaskuri

Juomatehtailija pullottaa limonadia $^1/_3$ litran pulloihin. Hän tarvitsee pullolaskurin, jolla voi laskea tarvittavan pullomäärän, kun ohjelmaan syötetään pullotettavan limonadin määrä litroina. Koodaa ohjelma, joka tulostaa tarvittavan pullomäärän, kun sille syötetään pullotettava limonadimäärä kokonaisina litroina. Ohjelma tulostaa täysinäisien pullojen määrän. Tallenna ohjelmasi KOODIT-kansioon nimellä *pullolaskuri.* Ohjelman toiminta on seuraava.

```
Lasketaan täysien 1/3 l pullojen määrä
Syötä pullotettava litramäärä: 100
Tarvittavien tyhjien pullojen määrä:  300
```

8. Yhtälöitä ja ongelmanratkaisua

Opit uutta:
- for-silmukan käyttö ongelmanratkaisussa
- yksinkertaisten matemaattisten sovellusten ohjelmointia

Osaat jo:
- if- ja elif -ehtolauserakenteet
- math- ja random kirjastojen käytön
- while- ja for-silmukat, range-lukujono

Tietokone on tehokas numeerisen tiedon käsittelyssä. Se suorittaa sekunnissa miljoonia laskutoimituksia, joten näillä ominaisuuksilla tietokone on mitä parhain matemaattisten ongelmien ratkaisijana. Tehtävien ratkaisussa käytetään ns. **raakaa voimaa** (*brute force*), jolloin ratkaisuehdotuksia testataan **silmukassa** ja mahdollinen ratkaisu tulostetaan näkyviin. Menetelmässä silmukalle valitaan riittävän laaja **lukualue**, josta ratkaisun arvellaan löytyvän.

Esimerkiksi yhtälön ***x+1 = 2x-7*** ratkaisun etsimisen *for*-silmukassa voi koodata oheisen esimerkin mukaisesti. Yhtälön tasapaino, eli ratkaisu (eli muuttujan x arvon) oletetaan löytyvän lukualueesta -100 – +100. Käydään tämä lukualue läpi for-silmukassa.

Yhtälö on 1. asteen yhtälö, jolla on mahdollisesti yksi ratkaisu tai ei yhtään ratkaisua. If-lauseessa testataan, onko yhtälön vasen puoli yhtä suuri kuin (==) oikea puoli. Jos ratkaiseva luku löytyy, niin tulostetaan ratkaisu ja lopetetaan silmukan suoritus break-lauseella.

Toisen ja korkeamman asteen yhtälöillä voi ratkaisuja olla useampia, eikä esimerkin mukainen ratkaisukeino ei ole riittävä. Pythonille on yhtälöiden käsittelyyn olemassa *sympy*-kirjasto, jota emme tässä yhteydessä käsittele.

valitaan range-lukualueeksi (-100,100), josta ratkaisun voi olettaa löytyvän

```
# yhtälön ratkaisua for-silmukassa
for x in range(-100,100):
        if x+1==2*x-7:
            print ("Yhtälön ratkaisu on: ",x)
            break
```

```
Yhtälön ratkaisu on: 8
```

Jos ratkaistava ongelma on esitetty sanallisessa muodossa, on tehtävästä muodostettava matemaattinen malli eli yhtälö. Matemaattinen yhtälö voidaan ratkaista tietokoneella. Seuraavassa esimerkissä sanallisesta ongelmasta muodostetaan yhtälö. Yhtälö koodataan ohjelmaksi ja etsitään ratkaisu.

Laurilla ja Eerolla on yhteensä 268 postimerkkiä. Kuinka monta merkkiä on kummallakin, kun Eerolla on merkkejä kolme kertaa niin paljon kuin Laurilla?

Laurilla on **x** kpl

Eerolla on **3x** kpl

Yhtälö **x + 3x = 268** → **4x = 268**

Tässä haetaan ensin ratkaisu Laurin postimerkkien määrälle, jonka jälkeen saadaan ratkaisu Eeron merkkien määrälle. Ohessa koodattu ratkaisu tehtävään.

```
# postimerkit
for x in range(0,268):
    if 4*x == 268:  # myös x+3*x==268 kelpaisi
        print ("Laurilla on merkkejä: ",x)
        print ("Eerolla on merkkejä: ", 3*x)
```

```
Laurilla on merkkejä:  67
Eerolla on merkkejä:  201
```

TEHTÄVIÄ

Tehtävä 29. Yhtälönratkaisua Pythonilla

Koodaa esimerkin mukainen yhtälön ratkaisuohjelma. Tallenna ohjelma KOODIT-kansioon nimellä *yhtälö.*

Ratkaise ohjelman avulla myös seuraavat yhtälöt. Muuta siis if-lauseen ehtoa yhtälöä vastaavaksi:

a. -2x = 8, x =

b. 8x = x + 63, x =

c. 4(x-3) = 2(x-4), x =

Tehtävä 30. Ongelmanratkaisu koodaamalla

Muodosta seuraavissa sanallisissa tehtävissä yhtälö. Koodaa yhtälö ohjelmaan ja etsi ratkaisu suorittamalla ohjelma.

a. mikä luku kolmella kerrottuna on 129?

V:

b. Vilmalla on rahaa 18 euroa enemmän kuin Minjalla. Kuinka paljon on kummallakin, kun heillä on yhteensä 98 euroa?

V:

Tehtävä 31. Sanallisia tehtäviä koodaamalla

Muodosta seuraavissa sanallisissa tehtävissä yhtälö. Koodaa yhtälö ohjelmaan ja etsi ratkaisu suorittamalla ohjelma.

a. Karilla on rahaa 16 euroa enemmän kuin Merjalla. Kuinka paljon on kummallakin, kun heillä on yhteensä 48 euroa?

V:

b. Jimi on kolme vuotta vanhempi kuin Anne, joka on viisi vuotta vanhempi kuin Sofia. Kuinka vanhoja lapset ovat, kun heidän yhteenlaskettu ikänsä on 28 vuotta?

V:

Tehtävä 32. Verrantoyhtälö koodaamalla

Verranto on kahden suhteen yhtäsuuruus. Verrantomuotoinen yhtälö voidaan ratkaista ohjelmoimalla. Koodattaessa verranto kirjoitetaan if-lauseen totuusehdoksi. Esimerkiksi verranto $\frac{x}{24} = \frac{5}{6}$ koodataan

if x/24 == 5/6

Lukualueeksi voidaan oheisessa esimerkissä valita 0-25, koska ratkaisun on oltava pienempi kuin 24

```
# verranto
for x in range(0,25):
    if x/24 == 5/6:
        print ("Ratkaisu on: ",x)
        break
```

```
Ratkaisu on:  20
```

Koodaa verrantomuotoinen yhtälön ratkaisuohjelma. Tallenna se KOODIT-kansioon nimellä *verranto*. Ratkaise ohjelman avulla seuraavat sanalliset tehtävät.

a. Luvun x suhde lukuun 36 on yhtä suuri kuin lukujen 2 ja 18 suhde.

x =

b. Auto kuluttaa polttoainetta 6 litraa 100 kilometrillä. Kuinka pitkä matka voidaan ajaa 40 litralla? Oletetaan, että kulutus pysyy samana.

V:

PERUSMATEMATIIKKAA OHJELMOIMALLA

matemaattisen ohjelmoinnin harjoituskirja 9-luokalle

Oppilaan nimi:_____

9-vuosiluokan tehtävät:

Python-ohjelmointia 9. vuosiluokan matematiikassa

Ohjelman luotettavuutta ja selkeyttä saadaan lisättyä aliohjelmilla. Aliohjelmia hyödyntämällä myös ohjelmoinnillinen (algoritminen) ajattelutapa toteutuu. Jakamalla ongelmaa pienempiin osiin aliohjelmiksi, saadaan ratkaistua suuretkin tehtävät.

9. vuosiluokan ohjelmointitehtävissä opitaan kehittämään ohjelmista helppokäyttöisiä ja hyödynnetään aliohjelmia, jolloin ohjelmista tulee selkeitä, helppoja laajentaa ja ylläpitää. Tehtävissä sovelletaan paljon matematiikkaa, jolloin samalla saat vahvistettua myös matematiikan taitojasi.

Tehtäväluettelo:

1. Käyttäjäystävällinen ohjelma, math-kirjasto
    1) Salasana
    2) Luvun neliö ja kuutio
    3) Ympyrän pinta-ala ja kehän pituus
    4) Tarkista syöte neliöjuuressa
2. Sovellustehtäviä, potenssit, binäärilukuja, Pythagoras
    5) Potenssilaskualgoritmi
    6) Kymmenlukuja ja binäärilukuja
    7) Kateetin ratkaisu
    8) Simo koodaa Pythagoraan kolmikkoja
3. Sovellustehtäviä, prosenttilaskuja
    9) Prosenttilukulaskuri
    10) Prosenttiarvolaskuri
    11) Muutosprosenttilaskuri
    12) Simon alennus/korotuslaskuri
4. Lista tietorakenteena
    13) Listaharjoitus
    14) Virheellinen paikkaindeksi
    15) Indeksointi takaperin
    16) Otetaan listasta viipale
5. Listametodit, listan käsittely silmukoissa
    17) Alkioiden syöttäminen listaan
    18) Alkioiden summa for-silmukassa
    19) Alkioiden keskiarvo
    20) Alkioiden vaihteluvälin pituus
6. Aliohjelmat
    21) Kolmen luvun summa ja erotus aliohjelmina
    22) Kolmen luvun keskiarvo aliohjelma
    23) Jaollisuustutkimus
    24) Simon K13 laskuri
7. Funktio-aliohjelma
    25) Suorakulmainen särmiö
    26) Suorakulmainen kolmio ja sen pinta-ala
    27) Lukulistan minimi, maksimi ja keskiarvo funktioina
    28) Kertoma n!
8. Avaruusgeometrian apuvälineet itse ohjelmoituna
    29) Tehtävän pääohjelma
    30) Ympyrälieriö
    31) Ympyräkartio
    32) Pallo
9. Tilastomatematiikkaa, keskiarvo, mediaani ja moodi
    33) Koulutodistus
    34) Jatkoa koulutodistukseen
    35) Tikkaohjelma
    36) Simon matematiikan kokeet
10. Klassinen todennäköisyys simuloituna
    37) Kivi, paperi, sakset
    38) Arpanopan silmäluvun todennäköisyys
    39) Silmälukujen todennäköisyysjakauma
    40) Kaksi arpanoppaa

1. Käyttäjäystävällinen ohjelma, math-kirjasto

Opit uutta:
- ohjelman suorittaminen ikuisessa while-silmukassa
- matemaattisten sovellusten ohjelmointia
- neliöjuurifunktio sqrt()

Osaat jo:
- Pythonin perusteet: muuttujat, vertailu, print, if-lause
- math-kirjastojen käytön
- while- ja for-silmukat, tyypinmuunnokset

Käyttäjäystävällinen ohjelma antaa käyttäjän valita, milloin ohjelman käyttö lopetetaan. Lopetuskyselyä varten ohjelmaan koodataan **ohjelmaluuppi**, jolloin ohjelman suoritus jatkuu, kunnes käyttäjä päättää sen lopettaa.

Pythonissa ohjelmaluuppi voidaan tehdä **ikuisella *while***-silmukalla. *While*-silmukka saadaan ikuiseksi asettamalla silmukan toistoehdoksi ***True***. Lopettamiskysely ohjelmoidaan ***if***-lauseeseen. Jos käyttäjä haluaa lopettaa ohjelman, lopetetaan ohjelmaluuppi ***break***-lauseella. Ohessa on yksinkertainen esimerkki ohjelmaluupista ja lopetuskyselystä. Tätä rakennetta voidaan käyttää yleisesti kaikissa ohjelmissa. **Huomaa, että ohjelmaluupin sisään tulevat lauseet pitää sisentää**. IDLE-editori tekee sisennykset automaattisesti.

```
# ohjelmaluuppi
lopetus = "e"
while True:
    # tähän koodataan suoritettava ohjelma
    lopetus = input("Haluatko lopettaa k/e? ")
    if lopetus == "k":
        break
print (”Heippa!”)
```

```
Haluatko lopettaa k/e? e
Haluatko lopettaa k/e? k
Heippa!
```

Voit käyttää tätä ohjelmaluuppia jatkossa harjoitustehtävissä.
Tallenna tämä koodi nimellä *pohja* KOODIT-kansioon. Voit avata pohjan aina uuden tehtävän pohjaksi.

Pythonin *math*-kirjaston avulla voidaan helposti ohjelmoida erilaisia matemaattisia sovelluksia. Kirjaston komennot saadaan käyttöön kirjoittamalla ohjelman alkuun rivi ***from math import ****

Math-kirjasto sisältää mm. seuraavat funktiot: **itseisarvo** (*fabs*), **suurin yhteinen tekijä** (*gcd*), **potenssifunktio** (*pow*), **neliöjuuri** (*sqrt*), **listan alkioiden summa** (*fsum*) ja likiarvon **pii** (*pi*).

Oheinen koodiesimerkki kysyy käyttäjältä kokonaisluvun, tulostaa luvun itseisarvon, neliön ja kuution sekä neliöjuuren. Lopuksi esitetään lopetuskysely.

```
# math-esimerkki
from math import *
lopetus = "e"
while True:
    luku = int(input("Anna kokonaisluku: "))
    print ("Luvun",luku,"itseisarvo:",fabs(luku))
    print ("Luvun",luku,"neliö:",pow(luku,2))
    print ("Luvun",luku,"kuutio:",pow(luku,3))
    print ("Luvun",luku,"neliöjuuri:",sqrt(luku))
    lopetus = input("Haluatko lopettaa k/e? ")
    if lopetus == "k":
        break
print ("Heippa!"
```

```
Anna kokonaisluku: 9
Luvun 9 itseisarvo: 9.0
Luvun 9 neliö: 81.0
Luvun 9 kuutio: 729.0
Luvun 9 neliöjuuri: 3.0
Haluatko lopettaa k/e? k
Heippa!
```

Jos syötteeksi annetaan negatiivinen luku, tulee tästä rivistä virheilmoitus. Osaatko selittää miksi?

TEHTÄVIÄ

Tehtävä 1. Salasana

Koodaa yksinkertainen salasanan tarkistusalgoritmi. Ohjelmassa on esiasetettu jokin merkkijono salasanaksi (esim. *salasana = "kaksoiskartio"*) Käyttäjää pyydetään syöttämään salasana. Jos salasana on oikein, ohjelma toivottaa käyttäjän tervetulleeksi. Ellei salasana ole oikein, sitä kysytään uudelleen.

Tallenna ohjelma KOODIT-kansioon nimellä *salasana*. Ohjelman toiminta on seuraava:

```
Anna salasana: hypotenuusa
Väärä salasana!
Anna salasana: pyramidi
Väärä salasana!
Anna salasana: kaksoiskartio
Salasana oikein, tervetuloa!
```

Miten tehtävän algoritmi mielestäsi täyttää turvallisen salasanan käsittelyn periaatteet?

_____

Miten algoritmia voisi kehittää, jotta se ei anna mahdollisuutta arvata salasanaa loputtomasti?

_____

Tehtävä 2. Luvun neliö ja kuutio

Koodaa ohjelma, joka tulostaa kokonaisluvun **neliön** ja **kuution**. Tuloksesta käyttäjän tulee arvata mikä luku on kyseessä. Koodaa ohjelma ikuiseen while-silmukkaan. Ohjelmassa on seuraavat toiminnot:

1. aloita ohjelma kirjoittamalla rivi *from random import **
2. arvotaan satunnaisluku väliltä 1-15 ja tallennetaan muuttujaan *luku*, *randint(1,15)*
3. näytetään arvotun luvun neliö ja kuutio
4. kysytään käyttäjältä *vastaus*
5. verrataan vastausta oikeaan tulokseen ja kerrotaan vastauksen tulos
6. kysytään lopetushalut

Tallenna ohjelma KOODIT-kansioon nimellä *neliokuutio*. Voit vaikeuttaa tehtävää laajentamalla satunnaislukualuetta. Ohjelman toiminta on seuraava:

```
Luvun neliö on 144 ja kuutio 1728
Mikä luku on kyseessä? 12
Oikein!
Haluatko lopettaa k/e? e
Luvun neliö on 225 ja kuutio 3375
Mikä luku on kyseessä? 15
Oikein!
Haluatko lopettaa k/e? k
```

Miten voisit geometrisesti havainnollistaa luvun *a* neliötä ja kuutiota? Piirrä ehdotuksesi.

a^2 :

a^3 :

Tehtävä 3. Ympyrän pinta-ala ja kehän pituus

Koodaa ikuisessa silmukassa toimiva ohjelma, joka kysyy käyttäjältä ympyrän säteen *r* ja laskee ympyrän kehän pituuden ja pinta-alan. Kehän pituus sekä pinta-ala tulostetaan. Ohjelma kysyy lopetuskyselyn. Tehtävässä tarvitaan *math*-kirjaston *pi()*-funktiota.

Pyöristä tulokset *round()*-funktiolla kahden desimaalin tarkkuuteen. Tallenna ohjelma KOODIT-kansioon nimellä *ympyrätehtävä*. Ohjelman toiminta on seuraava:

```
Lasketaan ympyrän ala ja kehän pituus
Anna ympyrän säde: 2
Kehän pituus on 12.57
2.0 -säteisen ympyrän ala on 12.57
Haluatko lopettaa k/e? e
Anna ympyrän säde: 100
Kehän pituus on 628.32
100.0 -säteisen ympyrän ala on 31415.93
Haluatko lopettaa k/e? k
```

Tehtävä 4. Tarkista syöte neliöjuuressa

Koodaa ohjelma, joka ikuisessa silmukassa laskee luvun neliöjuuren. Jos käyttäjä antaa negatiivisen luvun, kerrotaan syötteen olevan epäkelpo juurrettavaksi. Tallenna ohjelma KOODIT-kansioon nimellä *neliojuuri*. Ohjelman toiminta on seuraava:

```
Lasketaan positiivisen luvun neliöjuuri
Anna juurrettava: -12
Luku ei saa olla negatiivinen
Haluatko lopettaa k/e? e
Anna juurrettava: 256
Luvun neliöjuuri on 16.0
Haluatko lopettaa k/e? k
```

2. Sovellustehtäviä, potenssit, binääriluvut ja Pythagoras

Opit uutta:
- potenssilaskujen ohjelmointia
- binäärilukujen ja 10-lukujen yhteyden
- Pythagoraan lauseen ohjelmoinnin

Osaat jo:
- ikuisen while-silmukan, for-silmukan
- tehtävissä vaadittavat matemaattiset perusteet
- Pythagoraan lauseen

x^y

Potenssilaskut voidaan Pythonissa ohjelmoida kahdellakin tavalla. Yksinkertaisin tapa on koodata potenssi aritmeettisella operaattorilla **. Esimerkiksi 5^2 koodataan Pythonilla 5**2. Toinen tapa on käyttää *math*-kirjaston *pow()*-funktiota. Jotta *math*-kirjasto saadaan käyttöön, on ohjelman alkuun kirjoitettava rivi *from math import **. Samainen potenssi 5^2 saadaan myös koodaamalla *pow(5,2)*.

```
# potensseja
from math import *
lopetus = "e"
while True:
    kanta = float(input("Anna kantaluku: "))
    exp = float(input("Anna eksponentti: "))
    print ("Tulostetaan potensseja eri tavoilla")
    print ("pow-funktio", pow(kanta,exp))
    print ("**-operaatio", kanta**exp)
    lopetus = input("Haluatko lopettaa k/e? ")
    if lopetus == "k":
        break
print ("Heippa!")
```

Binääri- eli kaksilukujärjestelmässä luvut esitetään ykkösinä ja nollina. Binääriluku muunnetaan kymmenjärjestelmän luvuksi luvun 2 potenssien avulla. Binääriluvun oikeanpuolimmainen bitti on vähiten merkitsevä luku (eksponentti 0) ja vasemmanpuolimmainen bitti on eniten merkitsevä (eksponentti bittien määrä – 1). Muunnoksessa lasketaan bitin tulo kantaluvun 2 painoarvon mukaisen potenssin kanssa. Esimerkiksi seuraavien binäärilukujen **1111** ja **101010** muunnokset ja arvot 10-järjestelmässä ovat seuraavat:

$\mathbf{1} \cdot 2^3 + \mathbf{1} \cdot 2^2 + \mathbf{1} \cdot 2^1 + \mathbf{1} \cdot 2^0 = 8 + 4 + 2 + 1 = 15_{10}$

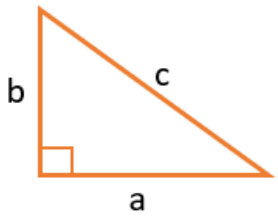
$\mathbf{1} \cdot 2^5 + \mathbf{0} \cdot 2^4 + \mathbf{1} \cdot 2^3 + \mathbf{0} \cdot 2^2 + \mathbf{1} \cdot 2^1 + \mathbf{0} \cdot 2^0 = 32 + 0 + 8 + 0 + 2 + 0 = 42_{10}$

(alaindeksi $_{10}$ tarkoittaa, että luku on kymmenjärjestelmän luku)

Muunnosalgoritmi olisi helposti Pythonilla toteutettavissa, mutta se on Pythonissa valmiina **tyyppimuunnosfunktiona *int(b,2)***, jossa *b* on kymmenluvuksi muunnettava binääriluku **merkkijonona** (ykkösinä ja nollina) ja parametri **2** määrää muunnoksen kantaluvusta 2 eli binääriluvusta. Esimerkiksi *int("101010",2)* antaa tulokseksi 42.

```
# binääriluku kymmenluvuksi
while True:
    b = input("Syötä bin.luku, esim 1010. e lopettaa: ")
    if b == "e":
        break
    else:
        print (int(b,2))
```

Pythagoras: $a^2 + b^2 = c^2$



$$c = \sqrt{a^2 + b^2}$$

$$a = \sqrt{c^2 - b^2}$$

$$b = \sqrt{c^2 - a^2}$$

Pythagoraan lauseella voidaan ratkaista suorakulmaisen kolmion sivun pituus, jos kaksi sivuista tunnetaan. Pythagoraan lauseella voidaan myös tutkia, onko kolmio suorakulmainen. Jos Pythagoraan lause $a^2 + b^2 = c^2$ pitää paikkaansa kolmion sivujen pituuksille, niin kolmio on suorakulmainen.

Suorakulmaisen kolmion sivujen pituus voidaan Pythagoraan neliöjuurilausekkeilla ratkaista Pythonissa *math*-kirjaston *sqrt* -funktion avulla. Oheisessa esimerkissä on ratkaistu hypotenuusan pituus, kun kateettien pituus tunnetaan.

```
# Pythagoras, hypotenuusa
from math import *
lopetus = "e"
while True:
    a = float(input("Anna kateetti a: "))
    b = float(input("Anna kateetti b: "))
    print ("Hypotenuusan pituus on:")
    print (sqrt(a**2+b**2))
    lopetus = input("Haluatko lopettaa k/e? ")
    if lopetus == "k":
        break
print ("Heippa!")
```

TEHTÄVIÄ

Tehtävä 5. Potenssilaskualgoritmi

Potenssilasku voidaan esittää tulona, jossa kantalukua kerrotaan itsellään eksponentin osoittama määrä. Esimerkiksi 5^3 voidaan esittää tulona $5 \cdot 5 \cdot 5 \cdot 5 \cdot 5 = 125$

Koodaa ohjelma, jossa kysytään kantaluku *k* ja eksponentti *x*. Ohjelman tulee laskea potenssin arvon joko **while**- tai **for**-silmukassa. Lopuksi potenssin arvo tulostetaan. Käytä ohjelmassa ikuista while-silmukkaa lopetusehdon kyselyssä.

Tallenna ohjelma KOODIT-kansioon nimellä *potensslilaskut*. Ohjelman toiminta on seuraava:

```
Lasketaan kokonaislukujen potensseja
Anna kantaluku: 3
Anna eksponentti: 4
3 potenssiin 4 on 81
Haluatko lopettaa k/e? e
Anna kantaluku: 10
Anna eksponentti: 6
10 potenssiin 6 on 1000000
Haluatko lopettaa k/e? k
```

Laske ohjelmasi avulla seuraavat luvun 2 potenssit:

2^0 = ____ 2^1 = ____ 2^2 = ____ 2^3 = ____

2^4 = ____ 2^5 = ____ 2^6 = ____

Tehtävä 6. Kymmenlukuja ja binäärilukuja

Kymmenluku voidaan muuntaa binääriluvuksi Pythonin tyypinmuunnosfunktiolla ***bin(n)***, jossa *n* on kokonaislukutyyppinen. Esimerkiksi *bin(21)* antaa tuloksen 0b10101. Pythonissa binääriluvun edessä on etuliite ***0b***.

Koodaa teoriaosassa esitetty binääriluvun muunnos kymmenluvuksi ohjelma. Lisää koodin loppuun toinen ikuinen while-silmukka, jossa muunnetaan käyttäjän antamia kymmenlukuja binääriluvuksi. Lisää myös samanlainen lopetusehto 'e'. Tallenna ohjema KOODIT-kansioon nimellä *bintodec.* Ohjelman toiminta näyttää seuraavalta:

```
Muunnetaan binääriluku 10-luvuksi
Syötä bin.luku, esim 1010. e lopettaa: 111
7
Syötä bin.luku, esim 1010. e lopettaa: 1111
15
Syötä bin.luku, esim 1010. e lopettaa: e
Muunnetaan 10-luku binääriluvuksi
Syötä 10-luku,e lopettaa: 22
0b10110
Syötä 10-luku,e lopettaa: 255
0b11111111
Syötä 10-luku,e lopettaa: e
```

Tee seuraavat lukumuunnokset. Alaindeksi $_2$ tarkoittaa binäärilukua ja $_{10}$ kymmenlukua:

1_2 = ____$_{10}$ 10_2 = ____$_{10}$ 100_2 = ____$_{10}$ 1000_2 = ____$_{10}$

10000_2 = ____$_{10}$ 100000_2 = ____$_{10}$ 1000000_2 = _____$_{10}$

Vertaa lukumuunnosten tuloksia edellisen tehtävän luvun 2 potensseihin. Mitä huomaat?

_____

Tehtävä 7. Kateetin ratkaisu

Koodaa ohjelma, joka suorakulmaisesta kolmiosta ratkaisee toisen **kateetin pituuden**, kun käyttäjä syöttää hypotenuusan ja toisen kateetin pituuden. Pyöristä tulos kahden desimaalin tarkkuuteen. Tallenna ohjelma KOODIT-kansioon nimellä *kateetti*. Ohjelman toiminta näyttää seuraavalta:

```
Ratkaistaan suorak. kolmion toinen kateetti
Anna toisen kateetin pituus: 12
Anna hypotenuusan pituus: 15
Hypotenuusan pituus on 15.0
Kateettien pituudet ovat 9.0 ja 12.0
Lopetetaanko k/e?: e
Anna toisen kateetin pituus: 1
Anna hypotenuusan pituus: 2
Hypotenuusan pituus on 2.0
Kateettien pituudet ovat 1.73 ja 1.0
Lopetetaanko k/e?: k
```

Miksi ohjelma antaa virheilmoituksen, jos hypotenuusan pituus on pienempi kuin kateetin pituus? _____

Tehtävä 8. Simo koodaa Pythagoraan kolmikkoja

Pythagoraan kolmikot ovat positiivisia kokonaislukuja, jotka täyttävät ehdon $a < b < c$ ja $a^2 + b^2 = c^2$. Ensimmäinen Pythagoraan kolmikon luvut ovat 3,4,5.

Alla on Simon yritys toteuttaa ohjelma, joka tulostaa pyydetyn määrän Pythagoraan kolmikoita. Simo on toteuttanut ratkaisun lähes oikein sisäkkäisillä *for*-silmukoilla. Koodissa on pieniä huolimattomuusvirheitä. Korjaa Simon koodi toimivaksi ja tallenna se KOODIT-kansioon nimellä *kolmikot*.

```
math import
n = input"Mikä on kolmikoiden yläraja c:lle"?
for a in range(1,n+1)
   for b in range(a+1,n+1)
      for c in range(b+1,n+1)
         if pow(a**2)+pow(b**2) == pow(c**2)
            print (a,b,c)
```

Mitkä Pythagoraan kolmikot löytyvät, kun c:n yläraja on 15?

_____

3. Sovellustehtäviä, prosenttilaskuja

Opit uutta:
- prosenttilaskujen ohjelmointi Pythonilla

Osaat jo:
- Pythonin perusteet
- prosenttilaskut, prosenttiarvo, muutosprosentti, alennus
- round()-funktion käytön pyöristämisessä

Pythonissa desimaalilukuja pyöristetään ***round(x,n)*** funktiolla, jossa *x* on pyöristettävä luku ja *n* on desimaalien määrä. Esimerkiksi *round(12.4567, 2)* pyöristyy likiarvoksi 12.46.

Pyöristäminen tehdään Python 3 versiossa ns. pankkiirin pyöristyssäännön mukaan.

Python tulostaa usein liikaa desimaaleja, jolloin esityksen selventämiseksi lopputulos on pyöristettävä haluttuun tarkkuuteen *round()* funktiolla.

Tehtävästä ja sen alkuarvoista riippuen lopputuloksen pyöristäminen tehdään pyöristyssääntöjen mukaan. Pyöristyssääntö voidaan karkeasti jakaa kahteen tapaan:

1. Yhteen- ja vähennyslaskuissa pyöristetään pyydettyyn tarkkuuteen tai mittayksikköön tai epätarkimman lähtöarvon mukaiseen tarkkuuteen.

2. Kerto- ja jakolaskuissa pyöristetään yhtä monen **merkitsevän numeron** tarkkuuteen kuin vähiten merkitseviä numeroita sisältävässä lähtöarvossa on.

Pyöristyssäännöt on vaikeaa ohjelmoida Pythonilla tai millään muullakaan ohjelmointikielellä. Esimerkiksi syötetyistä lukuarvoista merkitsevien numeroiden määrän selvittäminen olisi monimutkaista. Tästä syystä lopputuloksen tarkkuus päätetään johonkin sopivaan tarkkuuteen riippumatta lähtöarvoista.

```
# pankkiirin pyöristys
lopetus = "e"
while True:
    a = float(input("Anna float-luku: "))
    print ("Kahden desim. tarkkuus",round(a,2))
    print ("Kolmen desim. tarkkuus",round(a,3))
    lopetus = input("Haluatko lopettaa k/e? ")
    if lopetus == "k":
        break
print ("Heippa!")
```

```
Anna float-luku: 12.4567
Kahden desim. tarkkuus 12.46
Kolmen desim. tarkkuus 12.457
Haluatko lopettaa k/e? k
Heippa!
```

Prosenttilaskuihin liittyy mm. seuraavia laskukaavoja, jotka on helppo koodata Pythonilla. Muista pyöristää vastaukset sopivaan tarkkuuteen.

- **prosenttiluku**, $\frac{osa}{kokonainen}$ = *desimaaliluku* → *desimaaliluku* · 100 = *prosenttiluku* %

  esim. kuinka monta %:a luku 5 on luvusta 25? $\frac{5}{25} = 0{,}20$ → 0,20 · 100 = 20%

- **prosenttiarvo**, $\frac{prosenttiluku\%}{100}$ · *perusarvo* = *prosenttiarvo*

  esim. paljonko on 35% 60 €:sta? $\frac{35\%}{100} \cdot 60€ = 0{,}35 \cdot 60€ = 21\ €$

- **muutosprosentti**, $\frac{muuttunut\ arvo - alkuperäinen\ arvo}{alkuperäinen\ arvo}$ = *desimaaliluku* → *prosenttiluku*%

  esim. Tuotteen hinta alenee 50€:sta 35€:on. Mikä on alennusprosentti?

  $\frac{35€-50\,€}{50\,€} = -0.3 \rightarrow -30\%$, tässä miinus etumerkki tarkoittaa *alennus*prosenttia.

TEHTÄVIÄ

Tehtävä 9. Prosenttiluku

Koodaa ikuiseen *while*-silmukkaan ohjelma, joka laskee ja tulostaa prosenttilukuja. Käyttäjältä kysytään (perusarvo) *kokonainen* ja *osa*. Annetut arvot tallennetaan *float*-lukuina muuttujiin. Ohjelma tulostaa prosenttiluvun yhden desimaalin tarkkuudella. Yhdistä tulostukseen mukaan % -merkki.

Tallenna ohjelma KOODIT-kansioon nimellä *prosenttiluku*. Ohjelman toiminta näyttää seuraavalta:

```
Lasketaan prosenttiluku
Anna luku, josta osa otetaan: 190
Mikä osa kokonaisesta? 1.9
Luku 1.9 Luvusta 190.0 on 1.0 %
Lopetetaanko k/e?: e
Anna luku, josta osa otetaan: 5245
Mikä osa kokonaisesta? 1000
Luku 1000.0 Luvusta 5245.0 on 19.1 %
Lopetetaanko k/e?: k
```

Miten ohjelmaa tulisi muuttaa, jos haluttaisiin laskea **promilleja** ‰?

Tehtävä 10. Prosenttiarvo

Koodaa ikuiseen *while*-silmukkaan ohjelma, joka laskee prosenttiarvon, kun ohjelmaan syötetään (perusarvo) *kokonainen* ja *prosenttiluku*. Annetut arvot tallennetaan *float*-lukuina vastaaviin muuttujiin. Ohjelma tulostaa prosenttiarvon yhden desimaalin tarkkuudella. Tässä yksikkönä voidaan käyttää euroja, joka katenoidaan tulostukseen mukaan.

Tallenna ohjelma KOODIT-kansioon nimellä *prosenttiarvo*. Ohjelman toiminta näyttää seuraavalta:

```
Lasketaan prosenttiarvoja rahamäärästä
Anna euromäärä, josta arvo lasketaan: 150
Anna prosenttiluku: 15
15.0 %:a rahamäärästä 150.0 on 22.5 €
Lopetetaanko k/e?: e
Anna euromäärä, josta arvo lasketaan: 900
Anna prosenttiluku: 22.5
22.5 %:a rahamäärästä 900.0 on 202.5 €
Lopetetaanko k/e?: k
```

Mieti huolella tulostuslauseessa merkkijonojen ja muuttujien asettelu, jotta tulostuksen saa yhdelle riville.

Tehtävä 11. Muutosprosentti

Koodaa ikuiseen *while*-silmukkaan ohjelma, joka laskee muutosprosentin, kun ohjelmaan syötetään arvot muuttujille *alkuperainen* ja *muuttunut*. Annetut arvot tallennetaan *float*-lukuina vastaaviin muuttujiin. Ohjelma tulostaa muutosprosentin yhden desimaalin tarkkuudella. Tulostukseen katenoidaan %-merkki.

Tallenna ohjelma KOODIT-kansioon nimellä *muutosprosentti*. Ohjelman toiminta näyttää seuraavalta:

```
Lasketaan muutosprosentteja
Anna alkuperäinen arvo: 20
Anna muuttunut arvo: 25
Muutos arvosta 20.0 arvoon 25.0 on 25.0 %:a
Lopetetaanko k/e?: e
Anna alkuperäinen arvo: 450
Anna muuttunut arvo: 225
Muutos arvosta 450.0 arvoon 225.0 on -50.0 %:a
Lopetetaanko k/e?: k
```

Mitä + ja – merkit tarkoittavat muutosprosenttiluvussa?

_____

Tehtävä 12. Simon alennus/korotusprosentti

Simo on yrittänyt koodata ohjelman, jolla voi laskea korotettuja ja alennettuja hintoja. Ohjelmaan syötetään hinta ja muutosprosentti. Jos hintaa korotetaan, niin muutosprosentti syötetään positiivisena lukuna. Alennusprosentti syötetään negatiivisena lukuna.

Ohjelman on tarkoitus toimia ikuisessa while-silmukassa. Ohjelman pitäisi tulostaa korotettu tai alennettu hinta. Simo on melkein onnistunut ratkaisussa, mutta koodissa on pieniä syntaksivirheitä. Korjaa Simon koodi ja tallenna se KOODIT-kansioon nimellä *Simonproslaskuri*.

```
# Simon prosenttilaskuri
print "Ohjelma laskee korotetun/alennetun hinnan"
print "Syötä korotusprosentti positiivisena ja alennusprosentti negatiivisena lukuna"
while true
   hinta = desim(input("Anna hinta: "))
   muutospros = desim(input("Anna muutosprosentti: "))
   kerroin = muutospros/100 # muutetaan %-luku desimaalikertoimeksi
   muutoskerroin = 1,0 + kerroin
   uusihinta = muutoskerroin x hinta
   print "Hinta muutoksen jälkeen:",uusihinta
   lopetus = input("Lopetetaanko k/e?: ")
   if lopetus = "k":
      break
```

LASKE OHJELMALLA SEURAAVIIN UUDET HINNAT:

16.50 € alennus -30%:_____

127 € korotus 20% :_____

90 € alennus 15% :_____

4. Lista tietorakenteena

Opit uutta:
- lista tietotyypin
- listan luonnin sekä alkioiden lisäämisen ja tulostamisen
- listan pituuden len(), viipaleen ottamisen listasta

Osaat jo:
- Pythonin perustietotyypit: int, float, string
- asetuslauseen
- input-lauseen ja tyypinmuunnoksen

Pythoniin on perustietotyyppien (boolean, integer, float) lisäksi määritettävissä muitakin tietotyyppejä (lista, tupla, sanakirja). **Lista** (*list*) tietotyyppi on hyödyllinen myös matematiikan sovelluksien kannalta. Lista voi olla tyhjä tai sisältää yksi tai useampia peräkkäisiä alkioita. Listaa voidaan käsitellä tarkoitukseen sopivilla komennoilla. Listaan voidaan lisätä alkioita, siitä voidaan poistaa alkioita ja alkioiden arvoja voidaan muuttaa.

Listan alkiot voivat olla mitä tyyppiä tahansa, kuten lukuja, merkkijonoja tai toisia listoja. Samaan listaan ei kuitenkaan kannata laittaa erityyppisiä alkioita, koska tällöin listan käsittelystä tulee hankalaa. Listan rakenne muistuttaa merkkijonoa (ks. 8-luokan kappale 1 s.21).

indeksi	0	1	2	3	4
arvosana = [8,6,9,10,7]	8	6	9	10	7

Listassa alkiot ovat peräkkäin. Listan alkioilla on 'paikkanumero' eli indeksi. Ensimmäisen alkion indeksi on 0, toisen 1, kolmannen alkion indeksi on 2 ja niin edelleen. Viimeisen alkion indeksi on arvosana-listassa on sama kuin listan pituus – 1, eli *len(arvosana) – 1.*

Lista luodaan kirjoittamalla listan tunnus, asetusoperaattori ja sen jälkeen alkioiden arvot hakasulkeisiin pilkuilla eroteltuna. Esimerkiksi ***arvosana = [8,6,9,10,7]*** luo listan, jonka tunnus eli viittaus on *arvosana* ja alkiot on kirjoitettu hakasulkeiden sisään. Nimiä sisältävä viisialkioinen merkkijonolista luotaisiin lauseella ***nimet = ["Kimmo","Virpi","Taina","Jussi","Heli"]***.

Lista voidaan luoda tyhjänä. Tällöin hakasulkeisiin ei kirjoiteta alkioita. Esimerkiksi ***pisteet = []*** luo tyhjän lista, jonka tunnus on *pisteet*. Tyhjään listaan alkioiden arvoja voidaan syöttää esimerkiksi *for*-silmukassa.

Alkioiden lisääminen listaan tehdään ***append()***-komennolla. Esimerkiksi *nimet.append("Timo")* lisää *nimet*-listan viimeiseksi alkioksi merkkijonon "Timo" ja listan pituus olisi tämän jälkeen 6. Listan pituus saadaan ***len()***-funktiolla.

```
# listat
arvosana = [8,6,9,10,7]
nimet = ["Kimmo","Virpi","Taina","Jussi","Heli"]
lista = [] # tyhjä lista
print (arvosana)
print (nimet)
print ("Listan nimet pituus:",len(nimet))
nimet.append("Timo")
arvosana.append(8)
print (arvosana)
print (nimet) # tehtävässä 13 koodaa tähän asti
```

```
[8, 6, 9, 10, 7]
['Kimmo', 'Virpi', 'Taina', 'Jussi', 'Heli']
Listan nimet pituus: 5
[8, 6, 9, 10, 7, 8]
['Kimmo', 'Virpi', 'Taina', 'Jussi', 'Heli', 'Timo']
```

Uusi alkio lisätään append-metodilla listan loppuun. Lisätyt alkiot näkyvät uusissa tulostuksissa. **Metodi**-sanaa käytetään aliohjelmasta, joka ei palauta arvoa vaan ainoastaan suorittaa jonkun toiminnon.

Yhden alkion arvo saadaan luettua viittaamalla listan indeksiin. Esimerkiksi ***arvosana[0]*** viittaa lukuarvoon 8 ja ***arvosana[4]*** vastaavasti lukuarvoon 7.

Listan alkion arvo muutetaan asetuslauseella viittaamalla muutettavan alkion indeksiin. Esimerkiksi asetus ***arvosana[0] = 10*** muuttaisi listan ensimmäisen alkion arvoksi luvun 10.

```
print (arvosana[0],arvosana[5])
arvosana[0] = 10
arvosana[5] = 10
print (arvosana)
```

```
8 8
[10, 6, 9, 10, 7, 10]
```

Tulostuksessa näkyy arvosana -listan ensimmäinen ja viimeinen alkio sekä listan muutetut arvot

TEHTÄVIÄ

Tehtävä 13. Listaharjoitus

Koodaa teoriasivun # listat listaesimerkkiohjelman alkuosa (10 riviä) ja tallenna se KOODIT-kansioon nimellä *listat*. Esimerkkikoodia käytetään tämän sivun muissakin tehtävissä.

Suorita ohjelma ja varmista sen toiminta.

Koodaa seuraavat lisäykset koodin loppuun. Suorita ohjelma uudelleen ja vastaa seuraaviin kysymyksiin?

arvosana[1] = 10
arvosana[4] = 10
nimet.append ("Sanni")
nimet[0] = "Janne"
print (nimet)
print (arvosanat)

Mikä on *arvosana*-listan pituus muutosten jälkeen? _____

Mikä on *nimet*-listan alkion *Sanni* paikkaindeksi? _____

Nimet-listasta katosi *Kimmo*. Selitä miksi?

_____

Arvosana lista on lisäysten jälkeen: [_____]

Älä sulje tiedostoa, vaan jatka seuraavaan tehtävään.

Tehtävä 14. Virheellinen paikkaindeksi

Listalle varataan tietokoneen muistista sen pituutta vastaava RAM-muistialue. Listan viittaukset (indeksoinnit) on kohdistettava tähän varattuun muistialueeseen.

Lisää alla esitetty koodirivi *listat*-ohjelmaasi, tallenna ja suorita.

print (nimet[7])

Mitä tapahtuu?

_____

Mikä virheilmoitus tulostuu?

_____

Osaatko selittää virheen syyn?

_____

Korjaa nimet-listan indeksi sellaiseksi, että nimi *Sanni* tulostuu viimeiselle riville.

Älä sulje tiedostoa, vaan jatka seuraavaan tehtävään.

Tehtävä 15. Indeksointi takaperin

Listan viimeisen alkion arvon saa indeksillä [-1], toiseksi viimeisen indeksillä [-2] jne. Lisää *listat*-koodin loppuun seuraavat tulostusrivit, tallenna ja suorita.

print (nimet[-1])
print (nimet[-2])
print (nimet[-3])

Mitä *print*-lauseet tulostavat?

_____

Lisää vielä seuraava *print*-lause

print (nimet[: : -1])

Mitä viimeisin *print*-lause tulostaa?

_____

Älä sulje tiedostoa, vaan jatka seuraavaan tehtävään.

Tehtävä 16. Otetaan listasta viipaleita

Listasta saadaan indeksien avulla ns. **viipale** eli osajoukko. Viipale saadaan indeksoimalla viipaleen alku- ja loppukohta. Loppukohta ei tule mukaan viipaleeseen. Esimerkiksi *print (arvosana**[0:3]**)* tulostaisi arvosanat *8,10,9*.

Miten saat tulostettua *nimet*-listasta neljä ensimmäistä nimeä? Lisää toimiva viipale koodiisi.

print (nimet[___ : ___])

Miten indeksoit tulostuslauseen, jotta saadaan tulostettua nimet *Taina Jussi, Heli*? Lisää toimiva viipale koodiisi.

print (nimet[___ : ___])

Lisää vielä seuraava tulostusrivi koodin loppuun.

print (nimet[: : 2])

Minkä säännön mukaan nyt listan alkiot tulostuvat?

_____

Tallenna ohjelma. Voit sulkea *listat*-tiedoston.

5. Listametodit, listan käsittely silmukassa

Opit uutta:
- muutamia listametodeja ja -funktioita
- listan tulostaminen for-silmukassa
- muutamia listametodeja ja -funktioita

Osaat jo:
- for-silmukan
- while-silmukan
- input-lauseen ja tyypinmuunnoksen

Pythonissa on listojen käsittelyyn valmiina useita erilaisia komentoja (metodeja ja funktioita, metodi suorittaa toiminnon ja funktio palauttaa jonkin arvon, vrt. funktiolaskin). Edellä esitettiin jo listan alkioiden määrän palauttava funktio ***len()***-funktio sekä alkion lisäävä ***append()***-metodi. Seuraavassa taulukossa on muutamia listametodeja ja niiden toiminta. Käytetään esimerkkinä listaa *luvut = [8, 6, 9, 10, 7, 8].*

Metodi	Toiminta	Esimerkki
max(luvut) min(luvut)	Palauttaa listan suurimman ja pienimmän alkion. Toimii myös merkkijonoilla.	max(luvut) palauttaa luvun 10
append(x)	Lisää alkion x listan loppuun.	luvut.append(5)
count(x)	Palauttaa x alkioiden lukumäärän listassa.	luvut.count(8) palauttaa 2
sort()	järjestää listan suuruus- tai aakkosjärjestykseen.	luvut.sort()
reverse()	Kääntää listan alkioiden järjestyksen.	luvut.reverse()
len(luvut)	Palauttaa listan pituuden.	len(luvut) palauttaa luvun 6

Oheisessa esimerkissä on kahden listan käsittelyä listametodeilla.

```
# listametodeja
luvut = [8,6,9,10,7,8]
nimet = ["Kimmo","Virpi","Taina","Jussi","Heli"]
print ("Pienin",min(luvut),"suurin",max(luvut))
print ("Kahdeksikkoja on",luvut.count(8))
nimet.sort()
print ("Aakkosjärjestys",nimet)
nimet.reverse()
print ("Käännettynä",nimet)
```

```
Pienin 6 suurin 10
Kahdeksikkoja on 2
Aakkosjärjestys ['Heli', 'Jussi', 'Kimmo', 'Taina', 'Virpi']
Käännettynä ['Virpi', 'Taina', 'Kimmo', 'Jussi', 'Heli']
```

Koska listassa alkiot ovat peräkkäin indeksoituna järjestyksessä, voidaan listaa käsitellä silmukassa. Esimerkiksi ikuisessa **while-silmukassa** voidaan syöttää alkioita haluttu määrä ja ohjelmaluuppi päätetään lopetuskyselyllä. Ohessa on esimerkki, jossa listaan lisätään lukuja (float). Jotta luvut tallentuvat listaan lukuna, on syötteen tyyppi muunnettava merkkityypistä float-tyypiksi.

Olemassa olevan, alkioita sisältävän listan läpikäyminen on luontevaa **for-silmukan** avulla. For-silmukka askeltaa automaattisesti listarakenteen läpi ja silmukan suoritus päättyy viimeisen alkion jälkeen. For-silmukkaan tutustuttiin 8-luokan tehtävissä, jossa silmukassa käytiin läpi merkkijono. Seuraavassa ohjelmaesimerkissä jatketaan while-silmukassa luodun lukulistan käsittelyä ja tulostetaan alkiot allekkain for-silmukassa. For-silmukassa voitaisiin laskea esimerkiksi alkioiden keskiarvo. Keskiarvon laskemisessa tarvittava alkioiden lukumäärä saadaan funktiolla *len(luvut)*. Tämä jää kuitenkin harjoitustehtäväksi.

```
# lukulista
luvut = [] # tyhjä lista
lopetus = "e"
while True:
    luku = float(input("Syötä luku: "))
    luvut.append(luku)
    lopetus = input("Haluatko lopettaa k/e? ")
    if lopetus == "k":
        break
print ("Luvut:",luvut) # koodaa tehtävässä 17 tähän asti
```

```
Syötä luku: 2.4
Haluatko lopettaa k/e? e
Syötä luku: 1.56
Haluatko lopettaa k/e? e
Syötä luku: 22.96
Haluatko lopettaa k/e? k
Luvut: [2.4, 1.56, 22.96]
```

```
# tulostetaan luvut allekkain
for i in luvut:
    print (i)
```

```
2.4
1.56
22.96
```

For-silmukka on tehokas listan läpikäymiseen.
Tässä koodissa jatketaan while-silmukassa syötetyn lukulistan käsittelyä.

TEHTÄVIÄ

Tehtävä 17. Alkioiden syöttäminen listaan

Koodaa teoriasivun # lukulista mallikoodi, jolla saadaan syötettyä lukuja *luvut*-listaan while-silmukassa. Tallenna ohjelmasi KOODIT-kansioon nimellä *lukulista*.

a. Lisää ohjelman loppuun listametodit, joka tulostaa syötetyn listan **pienimmän** ja **suurimman** alkio.

b. Lisää ohjelman loppuun listametodi, joka tulostaa **listan alkioiden määrän** eli listan pituuden.

c. Tallenna ohjelma uudelleen.

d. Suorita ohjelma. Syötä **viisi** kokonais- tai desimaalilukua:

e. Mikä on listan pienin alkio?_____

f. Mikä on listan pituus?_____

Älä sulje tiedostoa, vaan jatka seuraavaan tehtävään.

Tehtävä 18. Alkioiden summa for-silmukassa

Jatka *lukulista*-ohjelmaa ja lisää loppuun for-silmukka, jossa lasketaan syötettyjen lukujen summa.

a. Lisää ennen *for*-silmukkaa muuttuja *summa* ja aseta sen alkuarvoksi 0 (*summa = 0).*

b. koodaa *for*-silmukka, jossa lista-alkiot läpi 'juokseva' silmukkamuuttuja ***i*** lisätään muuttujaan summa ja asetetaan uudelleen *summa*-muuttujan arvoksi (*summa = summa+i*).

c. Lopuksi summa tulostetaan ja yhdistetään mukaan myös selitys (esim. *Listan lukujen summa on: 57.2*).

d. Tallenna ohjelma uudelleen ja testaa toiminta.

Älä sulje tiedostoa, vaan jatka seuraavaan tehtävään.

Tehtävä 19. Alkioiden keskiarvo

Jatka lukulista-ohjelmaa ja lisää loppuun algoritmi, jossa lasketaan syötettyjen lukujen aritmeettinen keskiarvo ($\frac{Lukujen\ summa}{lukujen\ määrä}$).

Vihje: käytä apuna edellisessä tehtävässä laskettua *summa*-muuttujaa sekä listan alkioiden lukumäärän palauttavaa *len(luvut)*-funktiota.

Tallenna ohjelma uudelleen ja testaa sen toiminta.

Syöttämieni lukujen keskiarvo on: _____

Älä sulje tiedostoa, vaan jatka seuraavaan tehtävään.

Tehtävä 20. Alkioiden vaihteluvälin pituus

Vaihteluvälin pituus on tilastollinen tunnusluku, joka saadaan suurimman ja pienimmän alkion erotuksesta. Tämän määrittämisessä tarvitaan Pythonin listametodeja *min()* ja *max()*.

Lisää *lukulista*-ohjelman loppuun rivit, jolla saadaan tulostettua syötettyjen lukujen vaihteluvälin pituus.

Tallenna ja testaa ohjelman toiminta muutosten jälkeen.

Voit sulkea tiedoston *lukulista*.

Lukujen vaihteluväli on: _____

6. Aliohjelmat

Opit uutta:
- aliohjelman ohjelmointi Pythonilla
- aliohjelmaparametrin välitys
- aliohjelman kutsuminen pääohjelmasta

Osaat jo:
- Pythonin perusteet
- aritmeettisen keskiarvon ja kokonaislukujen jaollisuussäännöt

Ohjelmien luotettavuutta, tehokkuutta ja yleiskäyttöisyyttä voidaan parantaa aliohjelmilla. **Aliohjelma** on ohjelman osa, joka suorittaa jonkin yksittäisen tehtävän. Aliohjelmaa kutsutaan pääohjelmasta aliohjelman **tunnisteella**. Aliohjelmasta voidaan siirtyä toiseen aliohjelmaan tai palata takaisin pääohjelmaan.

```
# aliohjelma 1
def tervehdys():
    print("Hei, hauska nähdä.")

# aliohjelma 2
def onnittelu(nimi):
    print("Onnea",nimi)

# pääohjelma
tervehdys()
onnittelu("Maija")
```

```
Hei, hauska nähdä.
Onnea Maija
```

Yleiskäyttöisellä aliohjelmalla saadaan ratkaisu kaikkiin samantyyppisiin tehtäviin. Esimerkiksi lajittelualiohjelmalla voidaan lajitella järjestykseen lukuja siinä missä nimiluetteloitakin. Aliohjelmista käytettävät nimitykset ovat varsin kirjavia. Aliohjelmista käytetään myös nimityksiä metodi, rutiini ja operaatio. Aliohjelmia on kahta eri tyyppiä, **funktioita** ja **proseduureja.** Funktion kutsu palauttaa aina arvon. Proseduuri ei palauta mitään arvoa, vaan se suorittaa jonkin toiminnon; esimerkiksi tulostuksen.

Aliohjelmat suositellaan kirjoitettavaksi ohjelman alkuun ja pääohjelma kirjoitetaan viimeiseksi. Ohjelman suoritus alkaa kuitenkin pääohjelmasta.

Aliohjelman määrittely eli **signatuuri** aloitetaan varatulla sanalla *def*, jonka jälkeen kirjoitetaan aliohjelman nimi ja mahdollinen parametriluettelo sulkeissa. Aliohjelman määrittelyrivi päättyy kaksoispisteeseen. Aliohjelmassa suoritettavat lauseet sisennetään lohkoksi samoin kuin esimerkiksi *if*-lauseessa. Sisennys tulee IDLE-editorissa automaattisesti.

def ali(x,y):

Aliohjelman määrittely aloitetaan varatulla sanalla *def*, aliohjelman tunniste *ali* ja aliohjelmalle välitettävät parametrit luetellaan sulkeissa *(x,y)*. Parametrit eivät ole pakolliset, sulkeet ovat. Määrittelyrivi päättyy kaksoispisteeseen :

```
# lukujen summa
def summa(a,b):
    print("Lukujen summa", a+b)

# lukujen erotus
def erotus(a,b):
    print("Lukujen erotus",a-b)

# pääohjelma
n,s = 10,5 #muuttujien arvot
summa(n,s)
erotus(n,s)

print("Aliohjelmat suoritettu")
```

```
Lukujen summa 15
Lukujen erotus 5
Aliohjelmat suoritettu
```

Aliohjelmaa **kutsutaan** pääohjelmasta tunnisteella. Esimerkiksi kutsu

ali(2,7)

kutsuu yllä määriteltyä aliohjelmaa *ali* ja välittää aliohjelman parametrille x luvun 2 ja parametrille y luvun 7. Aliohjelman lohkossa parametreja x ja y (eli kutsun jälkeen lukuja 2 ja 7) voidaan käyttää esimerkiksi aritmeettisissa operaatioissa.

Aliohjelman suorituksen jälkeen kontrolli palaa pääohjelmaan. Proseduurissa kontrolli palaa aliohjelman kutsun jälkeen seuraavalle riville.

Ohessa esimerkki aliohjelmista, joissa tulostetaan kahden luvun summa ja erotus. Aliohjelman jälkeen tulostetaan pääohjelmassa lopetusilmoitus.

TEHTÄVIÄ

Tehtävä 21. Kolmen luvun summa ja erotus aliohjelmina

Koodaa teoriasivun malliohjelman # lukujen summa mukaisesti ohjelma, joka:

- pääohjelmassa kysytään (input) käyttäjältä kolme kokonaislukua (int) ja tallennetaan ne **kokonaislukuina** muuttujiin *n* , *s* ja *t*.
  (Muista muuttaa näppäinsyötteen tyyppi int-tyypiksi
- lisää aliohjelmiin kolmas parametri *c*, jotta kolmas luku saadaan mukaan laskutoimituksiin.
- muokkaa myös tulostuslausekkeiden aritmeettiset operaatiot, jotta tulostukset saadaan oikein
- muokkaa pääohjelmassa olevat kutsut siten, että myös kolmas parametri *t* välitetään aliohjelmaan
- tallenna ohjelma KOODIT-kansioon nimellä *proseduurit*

Ohjelman toiminta näyttää seuraavalta:

```
Anna kokonaisluku: 34
Anna toinen kokonaisluku: 21
Anna kolmas kokonaisluku: 12
Lukujen summa 67
Lukujen erotus 1
Aliohjelmat suoritettu
```

Älä sulje tiedostoa, vaan jatka seuraavaan tehtävään.

Tehtävä 22. Kolmen luvun keskiarvo aliohjelma

Määrittele *proseduuri* ohjelmaan *keskiarvo* **aliohjelma**, joka laskee kolmen luvun *a,b* ja *c* keskiarvon ja tulostaa sen aliohjelmassa.

Lisää myös pääohjelmaan aliohjelmakutsu *keskiarvo(n,s,t)*. Tallenna ohjelma uudelleen ja testaa sen toiminta. Ohjelman toiminta näyttää seuraavalta:

```
Anna kokonaisluku: 111
Anna toinen kokonaisluku: 222
Anna kolmas kokonaisluku: 333
Lukujen summa 666
Lukujen erotus -444
Lukujen keskiarvo 222.0
Aliohjelmat suoritettu
```

Tehtävä 23. Jaollisuustutkimus

Aloita uusi tiedosto ja koodaa aliohjelma *jaolliset(luku)*, joka tulostaa kaikki luvulla *luku* jaolliset luvut väliltä *luku* – 100. Toteuta tulostus aliohjelmassa toistolauseella eli silmukalla (*for* tai *while*). (Vihje: lause *x % 2 == 0* antaa tuloksen *True*, jos *x* on jaollinen luvulla 2)

Toteuta pääohjelma ikuisella *while*-silmukalla, jossa on lopetuskysely (ks. esimerkiksi T1). Pääohjelmassa käyttäjältä kysytään kokonais*luku*. Pääohjelmasta kutsutaan aliohjelmaa *jaolliset(luku)*, jolloin parametri välitetään aliohjelmalle.

Tallenna ohjelma KOODIT-kansioon nimellä *jaollistutka*. Testaa ohjelman toiminta, jonka pitäisi näyttää seuraavalta:

```
Anna luku: 25
Tulostetaan kaikki 25 jaolliset 100:n asti
25
50
75
100
Lopetetaanko? k/e e
Anna luku: 30
Tulostetaan kaikki 30 jaolliset 100:n asti
30
60
90
Lopetetaanko? k/e k
```

Tehtävä 24. Simon K13 laskuri

Simo on yrittänyt koodata aliohjelmia, mutta siinä on paljon virheitä. Pääohjelmassa kysytään käyttäjän nimi ja syntymävuosi sekä kuluva vuosi. Tervehdys tulostetaan aliohjelmassa *tervehdys*, jolle välitetään *nimi* parametrina. Aliohjelma *ikaraja* saa parametreikseen *svuosi* ja *nyt.* Aliohjelmassa lasketaan käyttäjän ikä. Jos ikä on alle 13, tulostetaan tästä viesti *"Et voi katsoa K13 elokuvia"*, muuten tulostetaan *"Voit katsoa K13 elokuvia"*

Korjaa Simon koodi ja tallenna se KOODIT-kansioon nimellä *simonlaskuri*.

```
# Simon K13 laskuri
sub tervehdys(nimi):
  print ("Hei",nimi)
sub ikaraja(svuosi, nyt):
  ika = svuosi - nyt
  if ika =! 13:
  print ("Et voi katsoa K13 elokuvia")
  else:
  print ("Voit katsoa K13 elokuvia")

# pääohjelma
while True:
  lopetus = "e"
  nimi = input("Kerro nimesi? ")
  nyt = int(input("Mikä vuosi on nyt? "))
  svuosi = int(input("Minä vuonna olet syntynyt? "))
  tervehdys()
  print ("Selvitetään ikäraja...")
  ikaraja()
  lopetus = input("Lopetetaanko? k/e ")
  if lopetus == "k":
    break
```

7. Funktio-aliohjelma

Opit uutta:
- funktion määrittelyn ohjelmaan
- funktion kutsuminen pääohjelmasta
- parametrien tarkoituksen aliohjelmissa

Osaat jo:
- aliohjelman määrittelyn
- Pythagoraan lauseen ja ympyrägeometrian
- listametodit min ja max

Funktio eroaa proseduurista siinä, että funktiossa on ***return***-lause, joka **palauttaa jonkin arvon** pääohjelmassa olevalle kutsulle. Ohjelmoinnillisesti eron voi ilmaista, että proseduurikutsu on **lause**, kun taas funktiokutsu on **lauseke**.

Aiemmin käsiteltiin esim. *math*-kirjaston funktioita *sqrt(), pow(), fabs()* jne., jotka palauttavat jonkin arvon. Koodaajan itse kirjoittamat funktiot toimivat samalla tavalla.

```
def fun(x,y):
        suoritettavat ohjelmarivit lohkona
        return arvo
# pääohjelmassa funktion kutsu
m = fun(a,b)
```

Pääohjelman funktiokutsussa välitetään parametrit *a* ja *b* funktioparametrien *x* ja *y* paikalle. Funktiossa suoritetaan ohjelmoidut operaatiot ja tuloksena saatu arvo palautetaan *return*-lauseella pääohjelmassa olevan kutsun *fun(a,b)* paikalle. Palautettu arvo sijoitetaan muuttujaan *m*.

Funktioiden avulla ohjelmakoodista tulee helposti luettavaa ja ylläpidettävää. Pääohjelmassa ovat funktiokutsut. Aliohjelmia ja funktioita voidaan tarvittaessa kirjoittaa lisää ja olemassa olevia muuttaa, eikä koko ohjelmaa tarvitse kirjoittaa uudelleen.

Funktioiden avulla saadaan helposti koodattua käyttökelpoisia aliohjelmakirjastoja matemaattisiin sovelluksiin. Esimerkiksi geometrian laskukaavat ja prosenttilaskukaavat voidaan kirjoittaa aliohjelmakirjastoiksi. Aliohjelmakirjaston funktioita voidaan käyttää eri ohjelmissa, eikä usein käytettäviä rutiineja tarvitse kirjoittaa aina uudelleen.

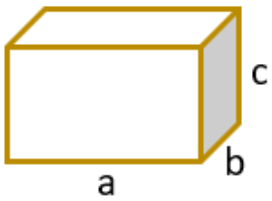
```
# summa-funktio
def summa(a,b):
    return(a+b)

# erotus-funktio
def erotus(a,b):
    return(a-b)

# tulo-funktio

def tulo(a,b):
    return (a*b)

# pääohjelma
n = float(input("Syötä luku n: "))
s = float(input("Syötä luku s: "))

# funktiokutsut
print(summa(n,s))
print(erotus(n,s))
print(tulo(n,s))
```

```
Syötä luku n: 3.5
Syötä luku s: 1.5
5.0
2.0
5.25
```

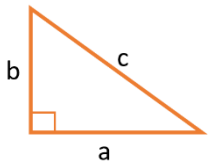
Tehtävä 25. Suorakulmainen särmiö

Ohessa on Simon lähes valmis funktioaliohjelmilla toteutettu ohjelma. Sillä saadaan laskettua suorakulmaisen särmiön tilavuus ja kokonaispinta-ala. Koodi on muuten valmis, mutta Simo ei muista miten tilavuus ja pinta-ala lasketaan eikä funktiokutsun syntaksia. Niihin kohtiin hän on kirjoittanut kysymysmerkkejä. Täydennä Simon koodi toimivaksi ja tallenna se KOODIT-kansioon nimellä *sksarmio*.

```
# suorakulmainen särmiö
def tilavuus(a,b,c):
   return(?????)

def kokpa(a,b,c):
   return (?????)

# pääohjelma
print ("""Lasketaan suorakulmaisen särmiön
tilavuus ja kokonaispinta-ala. Anna mitat""")
l = float(input("leveys: "))
p = float(input("pituus: "))
k = float(input("korkeus: "))
# funktiokutsut
print("Särmiön tilavuus on",?????)
print(" ja kok. pinta-ala",?????)
```



Tehtävä 26. Suorakulmainen kolmio ja sen pinta-ala



Koodaa funktioaliohjelmia hyödyntävä ohjelma, joka:

- sisältää seuraavat aliohjelmat
  - *onkosuorakulmainen(a,b,c)* , jossa parametrit a ja b ovat kateettien pituudet ja c on hypotenuusan pituus. Funktio palauttaa arvon *True*, jos kolmio on suorakulmainen, muuten palautetaan *False*. (Vihje: Pythagoraan lause)
  - *piiri(a,b,c)* , joka palauttaa kolmion piirin pyöristettynä kahden desimaalin tarkkuuteen
  - *pinta_ala(a,b)* , **jos kolmio on suorakulmainen**, niin funktio palauttaa suorakulmaisen kolmion pinta-alan pyöristettynä kahden desimaalin tarkkuuteen.

- pääohjelmassa kysytään kolmion sivujen pituudet *a, b* sekä *c* ja tallennetaan ne float-tyyppisinä vastaaviin muuttujiin
- pääohjelmassa tulostetaan kolmion piiri
- pääohjelmassa tarkistetaan, onko kolmio suorakulmainen. Jos on, niin lasketaan sen pinta-ala funktion avulla ja tulostetaan tulos pääohjelmassa

Tallenna ohjelma KOODIT-kansioon nimellä *kolmiolaskut* ja testaa ohjelma useilla eri syötteillä. Ohjelman toiminta näyttää seuraavalta:

```
Lasketaan kolmion piiri ja, jos se on
suorakulmainen, niin myös pinta-ala. Syötä sivujen pituudet
a: 3.0
b: 4
c: 5.0
Kolmion piiri 12.0
Pinta-ala 6.0
```

TAI

```
Lasketaan kolmion piiri ja, jos se on
suorakulmainen, niin myös pinta-ala. Syötä sivujen pituudet
a: 5
b: 6
c: 7
Kolmion piiri 18.0
Pinta-alaa ei voida laskea näillä tiedoilla
```

Miksi kolmion pinta-alaa ei voida laskea jälkimmäisen vaihtoehdon mukaisilla tiedoilla?

_____

Tehtävä 27. Lukulistan minimi, maksimi ja keskiarvo funktioina

Määrittele ja toteuta seuraavalle pääohjelmalle funktiot *pienin, suurin* ja *keskiarvo*. Pääohjelmassa lukuja syötetään silmukassa haluttu määrä. Funktiokutsussa *luvut*-lista välitetään aliohjelmalle. Tallenna toimiva ohjelma KOODIT-kansioon nimellä *listafunktiot.*

```
# pääohjelma
luvut = [] # tyhjä lista
lopetus = "e"
while True:
   luku = float(input("Syötä luku: "))
   luvut.append(luku)
   lopetus = input("Haluatko lopettaa k/e? ")
   if lopetus == "k":
      break
print ("Listan pienin luku on",pienin(luvut))
print ("Listan suurin luku on",suurin(luvut))
print ("Lukujen keskiarvo on",keskiarvo(luvut))
```

Tehtävä 28. Kertoma *n!*

Luvun **kertoma** (matemaattinen symboli **!**) on kaikkien lukua **n** pienempien tai yhtä suurien positiivisten **kokonaislukujen** tulo ja merkitään ***n!***
Esimerkiksi
3! = 1 · 2 · 3 = 6
5! = 1 · 2 · 3 · 4 · 5 = 120
Lisäksi on määritelty, että 0! = 1

Koodaa funktio *kertoma(n)*, joka laskee while-silmukassa **kokonaisluvun** n kertoman ja palauttaa kertoman arvon. Muista ohjelmoida myös ehto, jos n = 0, niin *n! = 1*. Koodaa myös pääohjelma, jolla funktion toiminta voidaan testata. Tallenna ohjelma KOODIT-kansioon nimellä *kertoma.*

Ohjelman toiminta on seuraava:

```
Minkä luvun kertoma lasketaan: 6
Luvun 6 kertoma on 720
```

8. Avaruusgeometrian apuvälineet itse ohjelmoituna

Opit uutta:

- vahvistat matemaattisten sovellusten ohjelmointia

Osaat jo:

- aliohjelmien määrittelyn
- funktioiden ja parametrien käytön
- avaruusgeometrian laskukaavat

Matematiikassa on paljon johdettuja valmiita laskukaavoja, joiden avulla saadaan useat tehtävät ratkaistua. Tämän harjoituksen tehtävissä kerrataan avaruusgeometrian kappaleisiin liittyviä laskuja. Palautetaan kappaleisiin liittyvät kaavat mieleen ja toteutetaan laskukaavat ohjelmoimalla.

Noudata matemaattisen ohjelman koodaamisessa seuraavia ohjeita:

- Jaa ohjelman toiminta pääohjelmaksi ja yhden tehtävän ratkaiseviksi aliohjelmiksi eli osatehtäviksi.
- Jaa osatehtävät mahdollisimman yksinkertaisiin osiin, joita hyödyntämällä saat ratkaistua suurempia tehtäväkokonaisuuksia.
- Matemaattisissa osatehtävissä funktiotyyppinen ratkaisu on yleensä hyödyllisin.
- Pääohjelmaan voidaan koodata alkuarvojen kyselyt käyttäjältä ja ohjelman käytön kannalta oleelliset valinnat.
- TÄRKEÄ! Ennen koodaamista, suunnittele ohjelman rakenne huolella. Mieti, mitä osia toteutetaan aliohjelmilla ja mitkä osat koodataan pääohjelmaksi.

AVARUUSGEOMETRIAN KAAVOJEN OHJELMOINTI

Ohjelmaa suunnitellessasi, mieti seuraavia asioita:

- Mitä lähtöarvoja käyttäjän pitää syöttää tehtävässä. Tallenna lähtöarvot muuttujiin.
- Mitkä osatehtävät ovat yhteisiä, ja ovat järkevää toteuttaa aliohjelmana / funktiona. Esimerkiksi sekä ympyrälieriössä ja ympyräkartiossa lasketaan pohjan pinta-ala. Se on viisasta toteuttaa aliohjelmana, jotta vältytään turhalta koodin kirjoittamiselta.
- Myös aliohjelmasta voidaan kutsua toista aliohjelmaa.
- Älä muuta käyttäjän antamia alkuarvoja (syötteitä) ohjelman suorituksen aikana. Jos arvoja on jostain syystä muutettava, niin kopioi muuttuja toiseen muuttujaan.

Alla on esimerkkiohjelma suorakulmaisen särmiön laskukaavojen toteutuksesta. Otetaan käyttöön *math*-kirjasto, jotta voidaan käyttää *pow*-funktiota. Vasemmalla on aliohjelmat, jotka palauttavat arvoja. Koodi jatkuu oikean puoleisessa ikkunassa, jossa on pääohjelma, funktiokutsut sekä lopetuskysely. Tutki koodi tarkoin.

```
from math import *
# suorakulmion ala
def sk_ala(a,b):
    return(a*b)

# kuution tilavuus
def kuutio_til(a):
    return(pow(a,3))

# kuution pinta-ala
def kuutio_ala(a):
    return (pow(a,2)*6)

# suorak särmiön tilavuus
def sarmio_til(l,s,k):
    # pohjan ala * korkeus
    return(sk_ala(l,s)*k)

# suorak särmiön pinta-ala
def sarmio_ala(l,s,k):
    # 2 * jokaisen tahkon ala
    return (2*(sk_ala(l,s)+sk_ala(l,k)+sk_ala(s,k)))
```

Kuution tilavuus ja pinta-ala eivät tarvitsisi omia aliohjelmia, mutta ne ovat tässä esimerkin vuoksi.

```
# pääohjelma
while True:
    print ("""Lasketaan suorakulmaisen särmiön
    tilavuus ja kokonaispinta-ala""")
    l = float(input("Syötä leveys: "))
    s = float(input("Syötä syvyys: "))
    k = float(input("Syötä korkeus: "))
    # funktiokutsut
    if (l==s and l==k):
        print ("Kyseessä on kuutio")
        print ("Sen pinta-ala on",kuutio_ala(l))
        print ("ja tilavuus",kuutio_til(l))
    else:
        print("Särmiön pinta-ala on",sarmio_ala(l,s,k))
        print("Särmiön tilavuus",sarmio_til(l,s,k))
#lopetuskysely
    lopetus = input("Jatketaanko k/e? ")
    if lopetus =='e':
        break
```

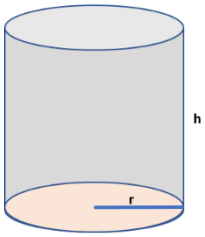
Tehtävä 29. Tehtävän pääohjelma

Tässä tehtäväsarjassa toteutat **avaruusgeometristen** kappaleiden ympyrälieriö, ympyräkartio ja pallo tilavuuksien ja pinta-alojen laskukaavat funktioina. Koodaa ensimmäiseksi oheinen pääohjelma ja tallenna se KOODIT-kansioon nimellä *ympyräkappaleet*. Selvitä pääohjelman toiminta rivi riviltä.

```
from math import *

# koodaa funktiot tähän väliin

# pääohjelma
while True:
    print ("Lasketaan ympyräkappaleen tilavuus ja pinta-ala")
    kappale = input("""Valitse kappale:
    l = ymp.lieriö, k = ymp.kartio, p = pallo """)

    if kappale == "l":
        r = float(input("Anna lieriön pohjan säde: "))
        h = float(input("Anna lieriön korkeus: "))
        print ("Lieriön tilavuus on",round(lieriön_tilavuus(r,h),2))
        print ("Lieriön vaipan pinta-ala on",round(lieriön_vaipan_ala(r,h),2))
        print ("Kok.pinta-ala on",round(lieriön_vaipan_ala(r,h) + 2*ympyrän_ala(r),2))

    lopetus=input("Haluatko lopettaa k/e? ")
    if lopetus == "k":
        break
```

Älä sulje tiedostoa, vaan jatka seuraavaan tehtävään.

Tehtävä 30. Ympyrälieriö



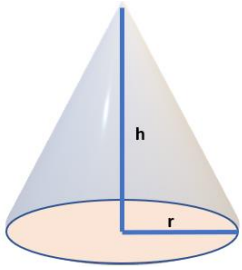
Koodaa edellisen tehtävän import-lauseen ja pääohjelman väliin **funktiot**, jotka **palauttavat** ympyrän alan, ympyrälieriön tilavuuden ja ympyrälieriön vaipan alan. Funktiokutsut on esitetty pääohjelmassa **lihavoituna**.

Tallenna ohjelma muutosten jälkeen ja testaa sen toiminta. Ohjelman pitäisi toimia seuraavasti:

```
Lasketaan ympyräkappaleen tilavuus ja pinta-ala
Valitse kappale:
l = ymp.lieriö, k = ymp.kartio, p = pallo l
Anna lieriön pohjan säde: 5
Anna lieriön korkeus: 10
Lieriön tilavuus on 785.4
Lieriön vaipan pinta-ala on 314.16
Kokonaispinta-ala on 471.24
Haluatko lopettaa k/e? k
```

Älä sulje tiedostoa, vaan jatka seuraavaan tehtävään.

Tehtävä 31. Ympyräkartio



Lisää pääohjelmaan ennen lopetuskyselyä vaihtoehdolle ”kartio” oheinen pääohjelmalohko (tai kopioi/liitä ”lieriö” osa). Koodissa on esitetty lihavoituna kartion tilavuuden, vaipan alan ja kokonaispinta-alan funktiokutsut. Koodaa funktioihin näiden arvot palauttavat lausekkeet.

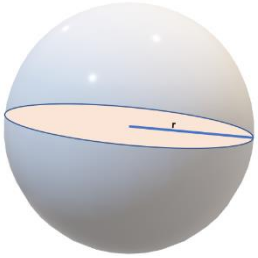
```
if kappale == "k":
    r = float(input("Anna kartion pohjan säde: "))
    h = float(input("Anna kartion korkeus: "))
    print ("Kartion tilavuus on",round(kartion_tilavuus(r,h),2))
    print ("Kartion vaipan pinta-ala on",round(kartion_vaipan_ala(r,h),2))
    print ("Kok.pinta-ala on",round(kartion_vaipan_ala(r,h) + ympyrän_ala(r),2))
```

Funktioiden lisäämisen ja koodin päivityksen jälkeen tallenna ohjelma uudelleen ja testaa sen toiminta. Kartion osalta ohjelman toiminta on seuraava:

```
Lasketaan ympyräkappaleen tilavuus ja pinta-ala
Valitse kappale:
l = ymp.lieriö, k = ymp.kartio, p = pallo k
Anna kartion pohjan säde: 5
Anna kartion korkeus: 10
Kartion tilavuus on 261.8
Kartion vaipan pinta-ala on 175.62
Kokonaispinta-ala on 254.16
Haluatko lopettaa k/e? k
```

Älä sulje tiedostoa, vaan jatka seuraavaan tehtävään.

Tehtävä 32. Pallo



Lisää pääohjelmaan kartion jälkeen ennen lopetuskyselyä oheinen pääohjelmalohko vaihtoehdolle ”pallo” tilavuuden ja pinta-alan laskemiseksi. Laskuja varten koodaa pääohjelmassa lihavoituna mainitut **funktiot** aliohjelmiin. Aliohjelmien tulee palauttaa kyseiset arvot pääohjelmaan.

```
if kappale == "p":
    r = float(input("Anna pallon säde: "))
    print ("Pallon tilavuus on",round(pallon_tilavuus(r),2))
    print ("Pallon pinta-ala on",round(pallon_ala(r),2))
```

Funktioiden lisäämisen ja koodin päivityksen jälkeen tallenna ohjelma uudelleen ja testaa sen toiminta. Pallon osalta ohjelman toiminta on seuraava:

```
Lasketaan ympyräkappaleen tilavuus ja pinta-ala
Valitse kappale:
l = ymp.lieriö, k = ymp.kartio, p = pallo p
Anna pallon säde: 20
Pallon tilavuus on 33510.32
Pallon pinta-ala on 5026.55
Haluatko lopettaa k/e? k
```

9. Tilastomatematiikkaa, keskiarvo, mediaani ja moodi

Opit uutta:

- listan tietojen käsittelyn
- statistics-kirjaston käytön tilastollisten tunnuslukujen esittämisessä

Osaat jo:

- lista-rakenteen Pythonissa
- tilastomatematiikan käsitteet; keskiarvo, moodi ja mediaani

Lista-rakenteeseen tutustuttiin tehtävässä 4. Listan alkioiden läpikäymisessä toistorakenteet eli for- ja while-silmukat ovat käyttökelpoisia.

Jos esimerkiksi listaan syötetään ennalta tunnettu määrä alkioita, voidaan syöttäminen toteuttaa **for** ja **range** -rakenteella.

While-silmukalla voidaan toteuttaa esimerkiksi järjestetystä listasta tietyn alkion etsiminen. Ohessa on esimerkkiohjelma listan alkioiden syöttämisestä ja tietyn alkion hakemisesta.

Esimerkkiohjelmassa luodaan tyhjä lista *luvut*, johon *for*-silmukassa lisätään lukuja *append*-metodilla. Lista tulostetaan. Käyttäjältä kysytään luku, jota etsitään *while*-silmukassa. Boolean muuttuja *löytyi* alustetaan arvoon *False*, jos lukua ei löydykään. Jos luku löytyy listasta indeksistä *paikka*, lopetetaan haku *break*-lauseella. Lopuksi tulostetaan hakutulos booleanmuuttujan *löytyi* ehdon mukaisesti.

```
# kokonaislukuja listaan
luvut = [] # tyhjä lista
lkm = int(input("Kuinka monta lukua syötät? "))
for i in range(0,lkm):
    luku = int(input("Syötä kokonaisluku: "))
    luvut.append(luku)
print ("Luvut:",luvut)
# alkion lineaarinen haku
haku = int(input("Mikä luku etsitään? "))
paikka = 0 # aloitetaan 0 indeksistä
löytyi = False
while paikka < len(luvut):
    if luvut[paikka]==haku:
        löytyi = True
        break
    else:
        paikka +=1 # indeksin kasvatus
if löytyi:
    print ("Luku löytyi indeksistä",paikka)
else:
    print ("Lukua ei löytynyt listasta")
```

```
Kuinka monta lukua syötät? 4
Syötä kokonaisluku: -21
Syötä kokonaisluku: 99
Syötä kokonaisluku: 32
Syötä kokonaisluku: -7

Luvut: [-21, 99, 32, -7]

Mikä luku etsitään? 99
Luku löytyi indeksistä 1
```

Numeerisen aineiston (datan) käsittelyyn Pythonissa on ***statistics*-kirjastomoduuli**. Kirjasto otetaan käyttöön kirjoittamalla ohjelman alkuun

*from statistics import **

Alla taulukossa on muutamia hyödyllisiä *statistics*-kirjaston funktioita.

mean()	aineiston arvojen aritmeettinen keskiarvo
median()	**järjestetyn** aineiston keskimmäinen arvo (tai kahden keskimmäisen arvon keskiarvo), mediaani. Listan voi järjestää *sort()* -metodilla, esim. *lista.sort()*
mode()	aineistossa eniten esiintyvä arvo, tyyppiarvo

Esimerkiksi ***mean([1, 2, 3, 4, 4])*** antaa tulokseksi 2.8 ***ja median([1, 3, 5])*** antaa tulokseksi 3.

Tämän harjoituksen tehtävissä ohjelmoidaan yksinkertaiset algoritmit tilastollisille tunnusluvuille: **keskiarvo**, **moodi eli tyyppiarvo**, **mediaani** sekä **vaihteluväli**. Vaihteluvälin toteuttamiseksi tarvitset listafunktioita *min()* ja *max()*. Myös **vaihteluvälin pituus** on helppo toteuttaa. Se saadaan aineiston suurimman ja pienimmän arvon erotuksena.

Tehtävissä voit alustaa listan arvot valmiiksi, jolloin niitä ei tarvitse syöttää aina uudelleen jokaisella suorituskerralla.

Tehtävä 33. Koulutodistus

Aloita tilastotehtävät koodaamalla ao. ohjelman aloitus ja tallenna se KOODIT-kansioon nimellä *koulutodistus.* Listoissa on muutama oppiaine ja niitä järjestyksessä vastaavat arvosanat. Voit lisätä halutessasi oppiaineita ja arvosanoja. Molempia on oltava yhtä monta, jotta ohjelman toimii oikein.

```
# oppiaineiden arvosanat
from statistics import *
aineet = ["MA","AI","FY","KE","BI","GE","HY","EA","RU"]
arvosanat = [9,8,9,7,10,8,8,7,8]
# tulostetaan aine ja arvosana
for i in range(0,len(aineet)):
    print (aineet[i],arvosanat[i])
```

Täydennä ohjelmaa siten, että se tulostaa arvosanojen **aritmeettisen keskiarvon** pyöristettynä kahden desimaalin tarkkuuteen. Pyöristäminen onnistuu *round()*-funktiolla.

Tehtävä 34. Jatkoa koulutodistukseen

Lisää *koulutodistus*ohjelmaan vielä arvosanojen **mediaanin** ja **moodin** tulostukset. Tulostukset voisivat näyttää seuraavalta.

```
MA 9
AI 8
FY 9
KE 7
BI 10
GE 8
HY 8
EA 7
RU 8
Arvosanojen keskiarvo 8.22
Arvosanojen mediaani on 8
Tyyppiarvo eli moodi on 8
```

Muuta arvosanoja ja testaa tulosten muuttuminen.

Tehtävä 35. Tikkaohjelma

Koodaa ohjelma, joka tallentaa listaan viiden heitetyn tikan osumat (0-10) *tikat*-listaan *for*-silmukassa, yksi arvo kerrallaan (vinkki: alusta nolla-arvot listaan *tikat = [0,0,0,0,0])* ennen for-silmukkaa. Lisäksi ohjelmassa lasketaan ja tulostetaan osumien keskiarvo (*mean()*), pienin (*min()*) ja suurin (*max()*) osuma. Lopuksi lasketaan ja tulostetaan vaihteluvälin pituus, joka saadaan suurimman ja pienimmän osuman erotuksena. Ohjelman suoritus näyttää seuraavalta:

```
1 .tikka
osuma: 1
2 .tikka
osuma: 7
3 .tikka
osuma: 10
4 .tikka
osuma: 6
5 .tikka
osuma: 5
Osumat: [1, 7, 10, 6, 5]
Keskiarvo: 5.8
Pienin: 1 Suurin: 10
Vaihteluvälin pituus 9
```

Tallenna ohjelma KOODIT-kansioon nimellä *tikkakisa.* Mitä vaihteluvälin pituus kertoo heittäjän taidoista?

_____

Tehtävä 36. Simon matematiikan kokeet

Simo on kiireessä tehnyt ohjelman, jolla hän haluaisi laskea matematiikan kokeiden aritmeettisen keskiarvon. Ohjelmaan Simo on yrittänyt määritellä kaksi listaa. Toisessa on kokeen aihe ja toinen taulukko sisältää samassa järjestyksessä kunkin aiheen arvosanan.

Simolla on aikomus tulostaa ensin vierekkäin riveittäin aihe ja koearvosana. Lopuksi tulostetaan kokeiden arvosanojen keskiarvo. Simo haluaisi ohjelman toiminnan olevan oheisen mallin mukainen.

Korjaa Simon koodi ja tallenna se KOODIT-kansioon nimellä *simonkokeet*.

```
Polynomit 9.25
Trigonometria 8.5
Geometria 10
Tilastot 7.75
Kertaus 8.75
-----------------
Keskiarvo 8.85
```

```
# Simon matematiikan kokeet
from random import *
koe = "Polynomit","Trigonometria","Geometria","Tilastot","Kertaus"
arvosana = 9.25, 8.5, 10, 7.75, 8.75
i = 0
while i == len(koe):
    print (koe[i], arvosana[i])
    i = 1
print("-----------------")
print ("Keskiarvo",mean(arvosana))
```

10. Todennäköisyys simuloituna

Opit uutta:

- satunnaisilmiön ohjelmointi Pythonilla

Osaat jo:

- Pythonin perusteet, for-silmukan
- random-kirjaston käytön ja funktiot randint ja choice
- Klassisen todennäköisyyden laskemisen

Tietokoneen laskentateho pääsee oikeuksiinsa erilaisissa simulaatioissa. Kone suorittaa sekunnissa kymmeniä tuhansia laskutoimituksia, joten simulaatioiden tulos saadaan melko nopeasti.

Pythonin *random*-kirjasto soveltuu hyvin erilaisten satunnaistapahtumien simulointiin eli mallintamiseen. Tällaisia simulaatioita ovat esimerkiksi nopan heitto, kolikon heitto, arvauspelit, korttipelit jne. Jotta *random*-kirjaston toiminnot saadaan käyttöön, tulee ohjelman alkuun kirjoittaa ***from random import ****.

Yhden arpanopan (silmäluvut 1-6) heitto voidaan simuloida funktiolla *randint(1,6)*, joka palauttaa satunaisluvun väliltä 1 – 6.

Useamman luvun lista saadaan funktiolla *sample(obj,k)*, joka valitsee k satunnaista alkiota listasta *obj*. Esimerkiksi lottorivi saadaan arvottua *sample(range(1,41),7)*, jossa *range*-funktion tuottamasta lukujonosta 1 – 41 poimitaan seitsemän satunnaista alkiota.

Kolikonheitossa on mahdollista saada arvot *kruuna* tai *klaava*. Tätä voidaan simuloida asettamalla mahdolliset arvot listaan; esim. *kolikko = ["kruuna","klaava"]*. Listasta voidaan poimia satunnainen alkio *choice()*-funktiolla, esimerkiksi *choice(kolikko)*.

```
# satunnaisuutta
from random import *
#lottonumerot, 7 numeroa väliltä 1-40
print (sample(range(1,41),7))
# kivi, paperi, sakset
lista=["kivi","paperi","sakset"]
print (choice(lista))
```

```
[3, 36, 10, 12, 29, 28, 19]
sakset
```

Klassinen todennäköisyys on suhdeluku, joka saadaan jakamalla suotuisten tapausten määrä *k* kaikkien tapausten määrällä *n.* Klassinen todennäköisyys voidaan esittää desimaalilukuna 0.0 – 1.0 tai prosenttilukuna 0% - 100%.

$$P(A) = \frac{k}{n}$$

Esimerkiksi arpanoppaa heitettäessä saadaan parillinen silmäluku todennäköisyydellä

$$\frac{3}{6} = 0.5$$

Todennäköisyyksiä voidaan tutkia toistamalla tapahtuma riittävän monta kertaa, jolloin toivotun tapahtuman (tilastollista) todennäköisyyttä voidaan verrata laskennallisesti saatuun tulokseen. Ohessa on esimerkkiohjelma, jossa simuloidaan yhden arpanopan heittoa ja yritetään saada silmäluku 4. Toistetaan heitto 100 kertaa. Tulosta voidaan verrata klassiseen todennäköisyyteen $\frac{1}{6} = 0.17$. Heittomäärällä 100 saattaa simuloitu tulos poiketa huomattavastikin laskennallisesta arvosta. Mutta lisättäessä heittojen määrää vaikkapa 100 000, niin simulaation tuottama tulos vastaa huomattavasti tarkemmin klassisen todennäköisyyden antamaa tulosta.

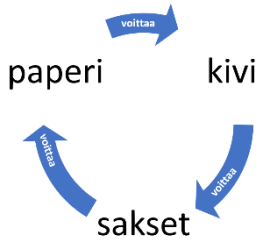
```
# arpanoppatutkimus
from random import *
heittoja = 100
suotuisia = 0 # alustus
haettu = 4
for i in range(heittoja):
    silmaluku = randint(1,6)
    if silmaluku == haettu:
        suotuisia += 1
print ("Silmälukua",haettu,"""
tuli""",suotuisia,"kappaletta")
print ("""Suotuisien suhde kaikkiin
heittoihin:""",suotuisia/heittoja)
```

```
Silmälukua 4
tuli 15 kappaletta
Suotuisien suhde kaikkiin
heittoihin: 0.15
```

Tehtävä 37. Kivi, paperi, sakset

Koodaa ohjelma, ikuisessa while-silmukassa kysyy toistuvasti ”kivi, paperi vai sakset?”. Ohjelmassa arvotaan *choice*-funktiolla listasta [”kivi”,”paperi”,”sakset”] satunnainen alkio. Käyttäjän antamaa arvausta verrataan arvottuun tulokseen ja tulostetaan arvauksen tulos. Lopetuskysely toteutetaan tavallisella lopetuskyselyllä.

Voittotestaus menee oheisen kaavion mukaan, jossa kivi voittaa sakset, sakset voittaa paperin ja paperi voittaa kiven. Yritä minimoida koodissa vertailujen määrä.



Tallenna ohjelma KOODIT-kansioon nimellä *kolikonheitto*. Testaa koodaamasi ohjelman toiminta, joka näyttää seuraavalta:

```
kivi, paperi vai sakset? kivi
Kone valitsi sakset
VOITIT!
Haluatko lopettaa? k/e e
kivi, paperi vai sakset? sakset
Kone valitsi paperi
VOITIT!
Haluatko lopettaa? k/e e
kivi, paperi vai sakset? sakset
Kone valitsi kivi
hävisit!
Haluatko lopettaa? k/e k
```

Vihje: testaa pelaajan valinnan mahdollinen voitto koneen valintaan verrattuna **kolmella** peräkkäisillä *elif*-vertailulla. Jos pelaajan valinta ei voita, niin tulosta *else*-haarassa lopuksi tieto häviöstä. Esimerkki vertailusta: *elif arvaus == "kivi" and kone == "sakset":*
print ("VOITIT!")

Tehtävä 38. Arpanopan silmäluvun todennäköisyys

Todennäköisyys valitulle arpanopan silmäluvulle on $\frac{1}{6} = 0.166\ldots$

Tutkitaan, millainen vaikutus heittojen lukumäärälle on todennäköisyyteen. Koodaa oheinen ohjelma, jossa for-silmukassa toistetaan ’nopan heitto’, eli toistuvasti arvotaan luku 1 – 6 ja lasketaan valitun silmäluvun esiintymien lukumäärät. Lopuksi tulostetaan todennäköisyys. Selvitä koodin jokaisen rivin tarkoitus ohjelman toiminnassa.

Tallenna ohjelmasi KOODIT-kansioon nimellä *arpanoppatutkimus*.

```
# arpanoppatutkimus
from random import *
heittoja = int(input("Kuinka monta heittoa? "))
suotuisia = 0 # alustus
valittu = int(input("Valitse silmäluku 1 - 6: "))
for i in range(heittoja):
    silmaluku = randint(1,6)
    if silmaluku == valittu:
        suotuisia += 1
print ("Silmälukua",valittu, "tuli",suotuisia,"kappaletta")
print ("Suotuisien suhde kaikkiin heittoihin:",suotuisia/heittoja)
```

Vaihtele heittojen lukumäärää ja kirjoita taulukkoon saatu todennäköisyys kullekin heittomäärälle:

heittomäärä	10	100	1000	10000
tod.näk				

Miten todennäköisyys muuttuu heittomäärän kasvaessa?

Tehtävä 39. Silmälukujen todennäköisyysjakauma

Kaikille arpanopan silmäluvuille todennäköisyys pitäisi olla sama, eli $\frac{1}{6} = 0.166\ldots$ Tutkitaan, pitääkö tämä paikkaansa kaikille 6:lle mahdolliselle silmäluvulle. Simuloidaan arpanopan miljoona heittoa ja lasketaan silmälukujen esiintymien lukumäärä. Lasketaan silmälukujen esiintymien suhteelliset määrät. Koodaa tutkimusta varten ohjelma, jolla saadaan selvitettyä silmälukujen todennäköisyys 1000 000 heitolla

Vihje: hyödynnä tässä kuusi alkioista taulukkoa, jossa ”ykköset” lasketaan indeksiin 0, ”kakkoset” indeksiin 1 jne. Lopuksi tulostetaan silmälukujen esiintymien todennäköisyydet. Kirjoita todennäköisyydet oheiseen taulukkoon:

silmäluku	1	2	3	4	5	6
tod.näk.						

Tallenna ohjelmasi KOODIT-kansioon nimellä *silmäluvut*.

Jakautuvatko silmälukujen todennäköisyydet tasaisesti:_____

Jos kyseessä olisi oikea noppa, eikä todennäköisyysjakauma olisi tasainen, mikä olisi johtopäätöksesi?

Tehtävä 40. Kaksi arpanoppaa

Mikä on todennäköisin silmälukujen summa heitettäessä kahta noppaa? Koodaa tätä tutkimusta varten noppaohjelma, jolla simuloidaan kahden nopan heitto esimerkiksi 10000 kertaa ja lasketaan noppien silmälukujen summien esiintymät. Luo noppien silmälukujen summia varten 11 alkioinen lista. Mahdollisia summia ovat luvut 2 – 12. Jos noppien summa on esimerkiksi 2 niin listan 0.-alkion arvoa kasvatetaan yhdellä ja vastaavasti summan ollessa 12 kasvatetaan listan 10. alkion arvoa yhdellä. Ohjelman lopuksi tulostetaan lista, josta voidaan päätellä useimmin esiintyvä noppien silmälukujen summa. Voit miettiä muitakin toteutustapoja tehtävän ratkaisemiseksi. Tallenna ohjelma KOODIT-kansioon nimellä *kaksinoppaa*.

Merkitse oheiseen taulukkoon silmälukujen summien esiintymät.

Σ	2	3	4	5	6	7	8	9	10	11	12
lkm											

Mikä luku osoittautuu todennäköisimmäksi kahden nopan summaksi? _____

Osaatko selittää, miksi yksi luku on todennäköisempi kahden nopan silmäluvun summana?