Yläkoulun ohjelmointia

Release 0.1.0

Tomi Pieviläinen

25.05.2016

Sisältö

genindex

LUKU 1

Valmiit oppitunnit

1.1 Ensimmäiset askeleet

1.1.1 Racketin ottaminen käyttöön

Tämän materiaalin ohjelmointiharjoitukset tehdään ohjelmassa **`DrRacket`**. Lataa omalle käyttöjärjestelmällesi sopiva versio Racketin sivuilta.

Kun DrRacket avataan ensimmäisen kerran, ohjelmointikieltä ei ole vielä valittuna. Tämän näkee alaosan punaisesta viestistä `No language chosen`. Kieli valitaan ylhäältä valikosta *Language* → *Choose language* tai painamalla `Ctrl-l`. Klikkaa tekstiä `Beginning Student` ottaaksesi kielen käyttöön. Kun olet sulkenut kielivalintaikkunan, paina vielä lopuksi ylhäältä `Run`-nappulaa.

Samasta valikosta voit myös vaihtaa kieltä myöhemmin, mikäli harjoitus niin vaatii.

1.1.2 Arvot ja funktiot

DrRacketissä on kaksi erilaista aluetta. Yläosaan kirjoitetaan kaikki tallennettava ja alaosaan voi laittaa nopeita kokeiluja. Yläosan koodi suoritetaan vasta, kun yläällä olevaa `Run`-nappulaa painetaan. Alaosassa taas riittää, kun painaa `Enter`.

Kokeile kirjoittaa alaosaan:

```
42
1/3
0.3
"Moi!"
true
false
```

Voit myös liittää kuvia leikepöydältä suoraan DrRackettiin.

Vihje: Saat kopioitua kuvan webbisivulta leikepöydälle klikkaamalla kuvaa oikealla hiiren näppäimellä (mäkeillä pitää klikata `Ctrl` pohjassa) ja valitsemalla valikosta `Kopioi kuva` tai vastaava komento (komennon tarkka kirjoitusasu riippuu selaimesta).

DrRacketissä voi liittää kuvia klikkaamalla oikealla hiiren näppäimellä ja valitsemalla `Paste`, painamalla `Ctrl+v` (mäkeillä `Cmd+v`) tai valikoiden kautta *Edit*→*Paste*.

Nämä kaikki olivat *arvoja*. DrRacket antaa aina jokaisen komennon jälkeen takaisin lopputuloksen arvon, joka tässä tapauksessa on se mitä itse kirjoitit.

Kokeile seuraavaksi näitä komentoja:

```
(+ 1 2)
(/ 1 3)
(string-length "Moi")
(string-length "Heippa, kaikki!")
(= 10 10)
(> 1/3 0.5)
```

DrRacket ei enää anna takaisin täsmälleen samaa, vaan se laskee annetun komennon arvon. Ohjelmoinnissa tällasia komentoja sanotaan *funktiokutsuiksi*. Ne alkavat aina sululla, jonka jälkeen tulee funktion nimi. Seuraavaksi niille annetaan välilyönnillä toisistaan erotettuja argumentteja, eli jotain tietoja, joita funktio käyttää. Viimeisen argumentin jälkeen tulee sulkeva sulku.

Argumentit ovat yksinkertaisimmillaan suoraan kirjoitettuja arvoja. Mutta koska funktiokutsujen lopputulos on aina arvo, argumenteiksi voi laittaa myös muita funktiokutsuja, jopa saman funktion kutsuja! Esimerkiksi jakolaskufunktion argumenttina voi olla toinen jakolasku.

Muista: Saat apua kaikista funktioista laittamalla kursorin funktion nimen päälle ja painamalla `F1`. Myös osa nimestä riittää, jos et esimerkiksi muista funktion koko nimeä. Jos käytät kannettavaa tietokonetta, voi olla että joudut painamaan myös `Fn`-näppäintä saadaksesi F1:n.

Arvot ovat aina jotakin *tyyppiä*. Ylempänä olevat esimerkit olivat *numeroita*, *merkkijonoja* eli tekstiä, *booleaneja* eli loogisen päättelyn vastaus kyllä tai ei (tosi ja epätosi) ja kuvia. Huomaa, että numero `42` ja merkkijono `"42"` ovat eri asioita! Ensimmäinen on matemaattinen käsite, toinen kaksi *merkkiä*.

Funktioiden argumenttien pitää olla tiettyä tyyppiä. Esimerkiksi koska `+` käsittelee numeroita, mutta ei tiedä mitä tehdä tekstille, `(+ 1 2)` toimii, mutta `(+ "12")` ei.

Rationaaliluvut voidaan ilmoittaa Racketissä kahdella lailla. Oletuksena DrRacket käyttää desimaalimuotoa, jonka toistuvan osuuden päälle piirretään viiva (eli Racket muistaa rationaaliluvut tarkasti). Samasta valikosta, mistä käytetty kieli valitaan, voi valita myös rationaalimuodon (*Language* → *Choose language* → *Show details* → *Fraction style* → *Mixed fractions*).

1.2 Laskujärjestys

Tätä kappaletta varten täytyy ensin lukea Ensimmäiset askeleet.

1.2.1 Peruslaskut Racketillä

Racketissä kaikki funktiokutsut, eli komennot, kirjoitetaan sulkujen sisään. Ensimmäiseksi tulee aina funktion nimi ja sen jälkeen välilyönnillä erotellut *argumentit* ja lopuksi sulkeva sulku.

Tehtävä

Mitkä seuraavista matemaattisista laskuista vastaavat mitäkin Racket-koodia?

```
(/ 2 (- 3 4))
(+ 2 3 4)
(* 2 (expt 3 4))
```

Tehtävä

Laske Racketillä:

Tehtävä

Keksitkö miten voisit laskea Racketillä

1.2.2 Moniosaiset laskut Racketillä

Edellisessä tehtävässä funktion *argumenttina* oli aina yksi luku ja yksi sellainen funktiokutsu, jonka lopputulos oli luku.

Esimerkiksi $5 + 15 : 3$ kirjoitetaan `(+ 5 (/ 15 3))` ja $10 : (5 - 3)$ `(/ 10 (- 5 3))`.

Yleisesti ottaen minkä tahansa luvun voi korvata funktiokutsulla ja funktiokutsuja voi olla sisäkkäin rajattomasti (tai oikeastaan rajoituksena on tietokoneen muistin määrä).

Tehtävä

Laske Racketillä

Tehtävä

Laske:

Tehtävä

Kokeile mitä tapahtuu, jos jätät alkavan sulun pois ennen komentoa. Entä lopettavan? Tai jos kirjoitat funktion nimen väärin tai annat liian vähän tai väärän tyyppisiä argumentteja?

Virheellisien komentojen kokeilu on hyödyllistä, sillä virheitä sattuu kaikille ohjelmoijille. Kun tiedät mistä virheestä tapahtuu mitäkin, osaat myöhemmin keksiä nopeammin mistä vahingossa tehty virhe johtuu.

1.2.3 Kiinnostuitko?

Kun painat `F1` jonkin laskufunktion päällä, pääset ohjeisiin, joista löydät myös kaikki muut mahdolliset matemaattiset funktiot, kuten eksponentin, neliöjuuren ja trigonometriset funktiot.

1.3 Piirtokomennot

Tätä kappaletta varten täytyy ensin lukea Laskujärjestys.

Jotta Racketillä voisi piirtää omia kuvia, ohjelman täytyy ladata käyttöön uusi *kirjasto* kirjoittamalla yläruutuun:

```
(require 2htdp/image)
```

Kirjaston lataaminen ei pelkästään vielä tee mitään, mutta se antaa käytettäväksi joukon uusia funktioita, kuten `circle`, `ellipse`, `line`, `rectangle` ja `triangle`. Esimerkiksi punaisen ympyrän saa piirrettyä funktiokutsulla:

```
(circle 20 "solid" "red")
```

Toinen *argumentti* voi olla joko `"solid"`, `"outline"` tai luku 0 ja 255 välillä.

Muista: Laita `require` funktion kutsu yläosaan ja paina `Run`, jotta saat piirtofunktiot käyttöön.

Tehtävä

Kokeile piirtää kaikilla yläpuolella mainituilla piirtofunktioilla.

Kokeile myös mitä tapahtuu, jos käytät `"solid"` sijasta numeroa.

Monimutkaisempia kuvia voi muodostaa asettamalla kuvia päällekäin funktiolla `overlay/offset`, jonka tulos on yksi yhdistetty kuva:

```
(overlay/offset (circle 20 "solid" "red")
                10 10
                (circle 30 "solid" "blue"))
```

Kokeile miten toinen ja kolmas *argumentti* vaikuttavat lopputulokseen!

Tehtävä

Osaatko paksuntaa reunaviivaa piirtämällä ympyröitä päällekäin tähän tapaan:

Tehtävä

Valitse kaksi eri maata ja piirrä niiden liput Racketillä.

Tehtävä

Piirrä peruskomentoja käyttäen yksinkertainen kuva autosta käyttäen alussa mainittuja piirtokomentoja.

Vinkki: kannattaa laittaa eri auton osat aluksi eri väreillä, jotta näet paremmin miten ne liikkuvat suhteessa toisiinsa. Monimutkaisempia kuvia kannattaa myös rakentaa ylempään ruutuun, jossa on helppo muokata aikaisempia käskyjä. Alemmassa ruudussa joudut joko kopioimaan ja liittämään vanhoja käskyjä, tai painamaan `Ctrl-nuoli ylös`.

1.3.1 Kiinnostuitko?

Voit saada lisää värejä käyttöön laittamalla viimeiseksi argumentiksi *merkkijonon* tilalle funktion `make-color` kutsun. Voit katsoa esimerkkejä kirjoittamalla `make-color`, laittamalla kursorin sen päälle, ja painamalla `F1`.

Katso myös mitä muita komentoja image.rkt -kirjastosta löytyy, sekä vilkaise piirtokomentojen opasta.

1.4 Vakiot

Tätä kappaletta varten täytyy ensin lukea Piirtokomennot.

Racketissä on mahdollista antaa nimiä eri arvoille. Tämä vähentää toistoa, jos arvo on laskettu pitkällä ja monimutkaisella funktiokutsulla. Se helpottaa myös ohjelman muuttamista: nimetty arvo tarvitsee muuttaa vain kerran, eikä jokaisessa paikassa, missä sitä käytetään. Nimeämistä sanotaan *määritelmäksi* ja nimettyä arvoa *vakioksi*. Nimi annetaan komennolla `define`:

```
(define nimi arvo)
```

Vakion nimi voi olla lähes mitä tahansa, mutta merkit (,), [,], {, }, ”, ‘, ‘, ,; ,# ,| ,\ ja itse pilkku ovat kiellettyjä. Lisäksi nimi ei saa näyttää numerolta.

Kokeile kirjoittaa DrRacketin yläosaan:

```
(require 2htdp/image)
(define ympyrät
  (overlay (circle 20 "solid" "red")
           (circle 30 "solid" "blue")))
```

Muista: Muista painaa `Run` aina sen jälkeen kun teet muutoksia yläosaan!

Muista: Jos et muista miten piirtokomennot toimivat, voit siirtää kursorin funktion nimen päälle ja painaa `F1`. Nettiselaimeen avautuu luettelo funktion nimellä löydetyistä ohjeista. Klikkaa 2htdp/imagen kohdalla olevaa linkkiä funktion ohjeisiin.

ja alaosaan:

```
ympyrät
```

Muista: Määritelmät täytyy kirjoittaa DrRacketin yläosaan, vaikka voitkin käyttää niitä sen jälkeen sekä ylhäällä että alhaalla!

`define` ei ole varsinaisesti funktio, sillä se ei palauta mitään (eli ei anna takaisin mitään, kun taas esimerkiksi `(+ 2 2)` tuottaa, eli palauttaa luvun 4). Jos siis esimerkiksi määrittelet kuvan, se ei tule näkyviin ennenkuin kirjoitat sille antamasi nimen (joko alaosaan, tai yläosaan jonka jälkeen täytyy painaa `Run`-nappia) ikäänkuin kirjoittaisit numeroita, merkkijonoja tai kopioisit kuvia. Vakioita voi laittaa myös funktioiden argumenteiksi, aivan kuten arvoja.

Muista: Huomaa, että vakio ei ole funktiokutsu, eli sen ympärille ei tule sulkuja! Se ei myöskään ole tekstiä, joten sen ympärille ei tule lainausmerkkejä. Eli jos vakio `pekka` olisi määritelty

```
(define pekka "kissanhiekka")
```

niin `pekka` antaisi sanan “kissanhiekka” ja `"pekka"` sanan “pekka”.

Tehtävä

Määrittele vakio, joka on kuva tikkarista.

Määrittele vielä eriniminen vakio erilaisesta tikkarista, ja kokeile molempia nimiä alaosassa.

Esimerkiksi kuvan metsästä voisi tehdä vaikka näin:

```
(require 2htdp/image)
(define latva
  (overlay/align/offset
   "middle" "top"
   (triangle 20 "solid" "green")
   0 10
   (triangle 30 "solid" "green")))
latva

(define runko (rectangle 10 10 "solid" "brown"))
runko
```

```
(define kuusi
  (above latva runko))
kuusi

(define metsä
  (beside kuusi kuusi kuusi kuusi kuusi))
metsä
```

Nyt jos kuusien väriä halutaan muuttaa, tarvitsee muutos tehdä vain vakion määritelmään:

```
(define latva
  (overlay/align/offset
   "middle" "top"
   (triangle 20 "solid" "darkgreen")
   0 10
   (triangle 30 "solid" "darkgreen")))
```

Tehtävä

Piirrä lehtipuu, anna sille nimi (eli määrittele vakio, jonka arvo on kuva lehtipuusta), ja piirrä metsä jossa on sekä kuusia, että lehtipuita.

Mitä vakioita määrittelit ratkaistaksesi tehtävän?

1.4.1 Kiinnostuitko?

Kokeile lisätä kuvaan taaemmaksi pienempiä puita, ikäänkuin ne olisivat kauempana. Tai vaikka sinisen taivaan, vuoren, lintuja jne.

1.5 Omat funktiot

Tätä kappaletta varten täytyy ensin lukea Vakiot.

Omat funktiot määritellään vakioiden tapaan kirjoittamalla

```
(define (funktion-nimi parametri toinen-parametri)
    ...)
```

Tässä suluissa oleva ensimmäinen "argumentti" näyttää funktiokutsulta, mutta ei oikeasti ole sitä, eikä edes argumentti: `define` on Rackettiin tehty poikkeustapaus (siksi se ei myöskään palauta mitään arvoa, kuten vakioihin tutustuttaessa huomattiin).

Pisteiden tilalle tulee joko arvo tai funktiokutsu, jonka lopputulos on myös uuden, tässä määritellyn funktion lopputulos. Sekä funktion että parametrien nimet saa päättää samoilla säännöillä kuin vakioidenkin nimet.

Parametrejä tulee olla vähintään yksi, mutta niitä ei ole pakko käyttää (tällöin on tosin parempi käyttää funktion sijaan vakiota). Ne toimivat niminä funktiokutsussa annetuille arvoille, ikäänkuin ne olisivat vakioita, jotka toimivat vain funktion sisällä.

Esimerkiksi funktio

```
(define (plus x y)
    (+ x y))
```

laskisi kahden annetun numeron summan. Se toimisi aivan samalla lailla, vaikka määrittelisit sen näin:

```
(define (plus koira kissa)
    (+ koira kissa))
```

eli annetuilla nimillä ei ole väliä, kunhan käytät samoja nimiä jotka olet antanut parametreille.

Tehtävä

Tee oma funktio, joka käyttää kolmea parametriä. Muista laittaa funktion ja parametrien nimien ympärille sulut, muuten Racket luulee sinun määrittelevän vakioita!

Funktio voi palauttaa mitä tahansa arvoja, ei pelkästään lukuja. Esimerkiksi

```
(define (pallo säde)
    (circle säde "solid" "yellow"))
```

palauttaisi annetun kokoisen, keltaisen pallon.

Kaikkia parametrejä ei ole pakko käyttää. Esimerkiksi

```
(define (funktio1 a b)
    2)
```

palauttaa aina arvon `2` riippumatta siitä, millä argumenteilla funktiota kutsutaan. Tosin tarpeettomista parametreistä ei ole mitään hyötyä.

Kuten vakiotkin, omat funktiot määritellään DrRacketin yläosaan, jonka jälkeen niitä voi käyttää kuten mitä tahansa muutakin funktiota.

Muista: Huomaa parametrin ja argumentin ero! Argumentti on tietoa, joka annetaan funktiolle sitä kutsuttaessa. Parametri on tälle tiedolle annettu nimi, jota funktio käyttää sisäisesti. Esimerkiksi ylempänä olevat a, b, x ja y ovat parametrejä, mutta funktiokutsussa `(plus 10 15)` 10 ja 15 ovat argumentteja.

Tehtävä

Tee funktiot `eur->usd` ja `usd->eur`, joilla voit muuttaa euroja dollareiksi ja toisinpäin.

Tehtävä

Tee funktio

- `kolmio`, joka antaa kuvan suorakulmaisesta kolmiosta, kun sille annetaan kateettien pituudet käyttäen valmista piirtofunktiota `right-triangle`
- `kolmion-pinta-ala`, joka laskee suorakulmaisen kolmion pinta-alan, kun sille annetaan kolmion kateettien pituudet.

- `hypotenuusa`, joka laskee hypotenuusan kun sille annetaan kolmion kateettien pituudet (neliöjuurifunktion nimi on sqrt).

Tehtävä

Tee funktio, joka piirtää annetun värisen auton (esim. `(piirrä-auto "red")`). Voit käyttää aiemman autotehtävän vastaustasi apuna.

Jos autosta tulee musta, Racket ei ole ymmärtänyt väriäsi. Ts. kaikki tuntemattomat värit oletetaan mustaksi. Muistele, miten vakioita ja parametrejä käytetään!

Tehtävä

Tee funktio, joka antaa toisen asteen yhtälön nollakohdan (toinen juurista riittää tässä vaiheessa), kun sille annetaan argumentteina polynomin kertoimet.

1.6 Ehtolauseet ja boolean-logiikka

Tätä kappaletta varten pitää ensin lukea Omat funktiot.

Liisa haluaa tilata pitsan. Hänellä on muutama ehdoton vaatimus: hän rakastaa homejuustoa, joten pitsassa täytyy olla aurajuustoa, ja lisäksi hän haluaa proteiinia, joten pitsa kelpaa vain jos siinä on joko kinkkua tai jauhelihaa. Ananakset ja persikat kuuluvat Liisan mielestä hedelmäsalaattiin, ei pääruokaan.

Eli Liisa suostuu syömään aura-kinkku-, aura-jauheliha- ja aura-kinkku-jauheliha-pitsan.

Tällaisia ehtoja sanotaan matematiikassa Boolen algebraksi. Siinä on pohjana kaksi totuusarvoa, tosi ja epätosi, ja kolme funktiota; `JA`, `TAI` ja `EI`. `JA` on tosi silloin kun kaikki sen argumenteista ovat tosia, `EI` jos ainakin yksi on tosi ja `EI` silloin kun sen ainut argumentti on epätosi.

Myös Racketissä on näille vastaavat funktiot. Liisan vaatimus Rackettinä olisi

```
(and onko-auraa
     (or onko-kinkkua onko-jauhelihaa)
     (not onko-ananasta)
     (not onko-persikkaa))
```

Tehtävä

Tee funktio, joka kertoo ovatko pitsan täytteet sellaiset, että sinä suostuisit syömään sen.

Jos olet erityisen kaikkiruokainen tai kranttu, voit myös käyttää kaverin toiveita.

Racketissä on myös komento, jolla erilaiset ehdot johtavat eri tuloksiin:

```
(cond
  [ehtolause tuloslause]
  [toinen-ehtolause toinen-tuloslause]
  [else kolmas-tuloslause])
```

Ehtolause on funktio, jonka arvoksi tulee joko tosi tai epätosi. Se voi käyttää hyväkseen Boolen funktioita, mutta myös esimerkiksi lukujen vertailuja. Tuloslause taas voi olla mitä tahansa.

Ehtolauseita voi olla miten monta vain ja lisäksi viimeisenä voi olla erikoistapaus `else`. `else` toteutuu aina, jos mikään edellinen ehtolause ei ole ollut tosi.

Lopuksi condin arvoksi tulee se tuloslause, jonka parina oleva ehtolause oli tosi.

Esimeriksi jos haluaisimme muuttaa lämpötilan asteina Celsiusta sanalliseksi kuvaukseksi, toimisi seuraavanlainen funktio:

```
(define (lämpötilan-kuvaus C)
  (cond
    [(> 40 C 25) "Liian kuumaa!"]
    [(> 25 C 18) "Kelvollinen sisälämpötila"]
    [(> 18 C 0 ) "Suomen kesä"]
    [(> 0 C -30) "Taitaa olla talvi"]
    [else "Oletko edes tällä planeetalla?"]))
```

Elseä tarvitaan, jos muut ehdot eivät kata kaikkia vaihtoehtoja. Jos ohjelman aikana käy niin, että mikään condin ehto ei ole tosi eikä sillä ole else-ehtoa, DrRacket antaa virheilmoituksen ja ohjelma pysähtyy.

Huomaa, että jos funktion lopputulos on boolean, ei condia tarvita lainkaan. Esimerkiksi funktio, joka kertoo syyssadetta inhoavalle onko sää sopiva ulkoiluun,

```
(define (voiko-mennä-ulos? C sataako?)
  (cond
    [(>= 30 C 22) true]
    [(>= 22 C 0) (not sataako?)]
    [(> 0 C -25) true]
    [else false]))
```

on mahdollista kirjoittaa tiiviimmin muodossa

```
(define (voiko-mennä-ulos? C sataako?)
  (or (>= 30 C 22)
      (and (>= 22 C 0)
           (not sataako?))
      (> 0 C -25)))
```

Tehtävä

Tee funktio, joka piirtää annetun värisen puun. Mutta väri annetaanki tällä kertaa suomeksi!

Laita funktio ymmärtämään ainakin kahta eri suomenkielistä väriä.

Muista myös miettiä mitä tapahtuu jos funktiota kutsutaan jollain muulla värillä, kuin mihin olit varautunut.

Muista: Condin ei tarvitse olla määritellyn funktion ensimmäisellä tasolla! Funktiossa voi käyttää condia argumenttina aivan kuin mitä tahansa muutakin funktiota, koska se palauttaa jotain. Esimerkiksi funktion, joka ymmärtää joitain suomenkielisiä värejä voisi tehdä näin:

```
(define (ympyrä väri)
  (circle 20 "solid"
          (cond
            [(string=? väri "keltainen") "yellow"]
            [(string=? väri "sininen") "blue"]
            [(string=? väri "punainen") "red"]
            [else "black"])))
```

1.6.1 Kiinnostuitko?

Boolen funktiot eivät ole ainoita loogisia funktioita. Funktioita voidaan esittää totuustaulussa. Taulun alussa on kaksi tai useampi argumentti, joiden arvo on joko tosi tai epätosi. Sen jälkeen listataan kullekin funktiolle onko se tosi vai epätosi samalla rivillä olevien argumenttien mukaan.

Tässä taulussa P ja Q ovat argumentteja, arvot on merkitty truen ja false'in etukirjaimilla.

P	Q	AND	OR	NAND |	NOR	XOR	IMP	EQV
T	T	T	T	F	F	F	T	T
T	F	F	T	T	F	T	F	F
F	T	F	T	T	F	T	T	F
F	F	F	F	T	T	F	T	T

Miten tekisit loput funktiot Racketissä olevilla `and`, `or` ja `not` funktioilla?

Tehtävä

Fibonaccin lukujono muodostuu näin:

- Ensimmäinen luku on 1
- Toinen luku on 1
- Muut luvut ovat kaksi edellistä lukua laskettuna yhteen

Tee funktio, joka ottaa yhden luonnollisen luvun N argumenttina ja palauttaa N:n Fibonaccin luvun.

Eli

```
(= (fib 1) 1)
(= (fib 2) 1)
(= (fib 3) 2)
(= (fib 4) 3)
```

ja niin edelleen.

Vihje: Tarvitset funktioon kolmen vaihtoehdon ehtolauseen. Ensimmäinen kertoo vastauksen, jos argumentti on 1. Toinen, jos 2. Kolmas laskee yhteen kaksi Fibonaccin-lukua. Millä funktiolla saat Fibonaccin luvun?

LUKU 2

Taustaa

2.1 Miksi opettaa tai opetella ohjelmointia?

Ohjelmointi on tulossa peruskoulun uuteen opetussuunnitelmaan kaikille luokka-asteille. OPS2016-dokumentti ei kuitenkaan perustele miksi ohjelmointia tulisi opettaa jokaiselle. Asiaa voi lähestyä monesta eri näkökulmasta, ja vertailla muihin oppiaineisiin.

2.1.1 Valmennus työelämään

Teollisuuden toiveissa on aina saada koulutusputkesta mahdollisimman valmiiksi koulutettua työvoimaa mahdollisimman suurissa määrissä. Kaikista ei jatkossakaan tule koodareita, mutta ala on jatkuvasti kasvava ja kaikki otsikoissa olevat kansantalouden suuret lupaukset ovat ohjelmistoalalla. Tällä hetkellä ohjelmoinnin opiskelu aloitetaan vasta korkeakoulutasolla. Jos jo aloittavat opiskelijat osaisivat koodauksen alkeet, he voisivat valmistua nopeammin tai ehtiä opiskelemaan alaa pidemmälle. Lisäksi monet saattavat jättää valitsematta alan, koska eivät oikeastaan tiedä mistä koko hommassa on kyse.

Tässä suhteessa ohjelmointia voisi verrata matematiikkaan, jonka alle OPS:n luonnos ohjelmoinnin onkin laittanut. Harva tarvitsee arki- tai työelämässään perusaritmetiikkaa pidemmälle menevää matematiikkaa, mutta silti jo peruskoulussa käydään läpi geometriaa ja algebraa. Tämä kuitenkin innostaa monia valitsemaan luonnontieteet tai teknologia-alan toisen asteen ja korkeakouluopinnoissaan, ja luo näille valmiin pohjan.

2.1.2 Yleinen ohjelmointitaito

Ohjelmoinnista voi silti olla hyötyä myös muille kuin ammatikseen koodaaville. Otso Kivekäs kirjoitti blogissaan tapaamastaan lääkäristä, joka pienellä itselleen tekemällä apuohjelmalla säästi aikaa asiakkaiden tietojen syöttämisestä tietokoneelleen. Lääkäri arvioi säästäneensä puolen vuoden aikana jo useamman työpäivän verran aikaa.

Samanlaisia pieniä tarpeita on vaikka missä. Liian pieniä, jotta työnantaja alkaisi virallisesti tekemään hankintaa, tai ettei kuluttaja viitsi käyttää aikaa etsien Internetin heinäsuovasta sopivaa appsia, jos sellaista on kukaan edes keksinyt tehdä. Tarpeeksi pieniä, että sen pystyisi tekemään amatöörikin, jos vain perustaidot löytyisivät.

Tähän tarpeeseen vertautuvat käsityöt. Siinä missä ennen riitti kun osasi parsia reiät ja irronneet napit, sekä kiinnittää naulakon ja nikkaroida mökille penkin, nykymaailmassa tarvitaan ohjelmointitaitoa arjen pieniin askareisiin.

2.1.3 Maailman ymmärtäminen

Koulussa on pitkään opetettu biologiaa, maantietoa, fysiikkaa ja kemiaa, jotta ymmärtäisimme maailmaa, jossa elämme. Mutta missä maailma oli ennen mekaaninen, se on nyt jatkuvasti enemmän elektroninen. Tietokoneohjelmat he-

rättävät meidät aamulla, tuovat uutiset aamupöytään ja ohjaavat liikennettä matkalla töihin.

Ne myöskin vievät yhä suuremman osan kuntien ja valtion budjeteista, mutta äänestäjillä ei oikeastaan ole käsitystä siitä mikä suurinpiirtein voisi olla mahdollista ja miten suurella vaivalla.

Ohjelmoinnin ja ohjelmistojen tekemisen perustaidot ovat nykyään merkittävä osa yleissivistystä.

2.1.4 Itsensä ilmaisu

Koulun tehtävänä ei ole tuottaa pelkästään hyviä työntekijöitä ja valveutuneita kansalaisia. Koulu antaa kaikille myös mahdollisuuksia ilmaista itseään musiikin, kuvaamataidon, äidinkielen ja käsitöiden kautta. Ohjelmointi voi olla osana kaikkea näitä, ja tuoda uusia mahdollisuuksia. Tarinankerronta ei ole enää yksipuolista kirjan lukemista tai elokuvan katsomista, vaan pelien maailmassa vastaanottaja imeytyy mukaan kertomukseen, samoin kuin musiikki ja visuaalinen näkymä reagoivat tilanteeseen. Vaikka pelit ovatkin arkipäiväisin ilmentymä taiteellisesta koodauksesta, monet ruutujen ulkopuolellakin olevat teokset ovat suurelta osin ohjelmakoodia.

Myös itse ohjelmakoodissa voi olla oma kauneutensa, ja monet ohjelmoijat arvostavatkin elegantteja, heidän silmiinsä taidokkaita ratkaisuja ongelmiin. Normaalin ammattiylpeuden lisäksi tämä voi ilmentyä myös omana taidelajinaan.

2.1.5 Ajattelutapojen oppiminen

Ohjelmointi vaatii ja harjoittaa joitain välttämättömiä taitoja, joita ei suoraan opeteta missään muussa oppiaineessa, mutta jotka silti ovat avuksi kaikessa muussa oppimisessa ja ongelmanratkaisussa. Ensimmäisenä vastaan tulee tietokoneen yksinkertaisuus: se tekee vain mitä käsketään, ja vain sen. Ihmiset kykenevät yleensä ymmärtämään myös toistensa epäselviä ohjeita, mutta väärinkäsitykset ovat silti yleisiä. Tietokoneet ovat anteeksiantamattomia epäselvyyksissä, ja pakottavat ohjelmoijan miettimään mitä hän tarkalleen haluaakaan.

Ohjelmointi opettaa myös pilkkomaan ison ongelman pienempiin paloihin ja tunnistamaan mitkä ongelmat ovat oikeastaan samoja, vain hieman eri lähtökohdista. Ohjelmien laajentuessa näistä pienistä paloista pitää myös muodostaa abstraktimpia toiminnallisuuksia, jotka tekevät enemmän, mutta vaativat kognitiivisesti yhtä vähän huomiota. Tämä on tärkeää, sillä kukaan ei kykene pitämään kovin suurta määrää pikkutarkkaa tietoa yhtäaikaa mielessään.

2.2 Miksi opettaa ohjelmointia tämän materiaalin avulla

Opettaa samalla matematiikkaa -> ei vie aikaa pois matematiikan opetukselta.

Opettaa joitain matematiikan käsitteitä helpommin/paremmin kuin paperilla.

Pohjautuu pitkään hiottuun ohjelmoinnin opetuksen tapaan.

Käyttää opetukseen suunniteltua kieltä ja ympäristöä.

Huomioi suomalaisen OPSin.

Testattu suomalaisilla opettajilla ja koululuokilla.

Avoin.

Päivittyy koko ajan, mutta ei muutu vain muuttumisen ilosta.

2.3 OPS 2016 luonnoksen ydinkohtia ohjelmoinnin kannalta

2.3.1 Laaja-alainen osaaminen vuosiluokilla 7-9

L5: Tieto- ja viestintäteknologinen osaaminen: — Ohjelmointia harjoitellaan osana eri oppiaineiden opintoja —

2.3.2 OPS2016 ehdotus 15.10.2014 päättyneellä lausuntokierroksella matematiikan opetukseksi yläkoulussa

Tavoitteet

Tavoitteen numero	Opetuksen tavoitteet	Tavoitteisiin liittyvät sisältöalueet	Laaja-alainen osaaminen, johon tavoite liittyy
Merkitys, arvot ja asenteet			
T1	vahvistaa oppilaan motivaatiota, positiivista minäkuvaa ja itseluottamusta matematiikan oppijana.	S10–S15	L1, L3, L5
T2	kannustaa oppilasta ottamaan vastuuta matematiikan oppimisesta sekä yksin että yhdessä toimien.	S10–S15	L3, L7
Työskentelyn taidot			
T3	vahvistaa oppilaan kykyä havaita ja ymmärtää oppimiensa asioiden välisiä yhteyksiä.	S10–S15	L1, L4
T4	harjaannuttaa oppilasta täsmälliseen matemaattiseen ilmaisuun suullisesti ja kirjallisesti.	S10–S15	L1, L2, L4, L5
T5	kehittää oppilaan taitoa ratkaista loogista ja luovaa ajattelua vaativia matemaattisia ongelmia.	S10–S15	L1, L3, L4, L5, L6
T6	ohjaa oppilasta arvioimaan ja kehittämään matemaattisia ratkaisujaan sekä tarkastelemaan kriittisesti tuloksen mielekkyyttä.	S10–S15	L1, L3, L4, L6
T7	rohkaisee oppilasta soveltamaan matematiikkaa muissakin oppiaineissa ja ympäröivässä yhteiskunnassa.	S10–S15	L1–L7
T8	kehittää oppilaan tiedonhallinta- ja analysointitaitoja sekä ohjata oppilasta tiedon kriittiseen tarkasteluun.	S10, S13, S15	L1, L4, L5
T9	soveltaa tieto- ja viestintäteknologiaa matematiikan opiskelussa sekä ongelmien ratkaisemisessa.	S10–S15	L5
Käsitteelliset ja tiedonalakohtaiset tavoitteet			
T10	vahvistaa päättely- ja päässälaskutaitoa sekä kannustaa oppilasta käyttämään laskutaitoaan eri tilanteissa.	S10, S11	L1, L3, L4
T11	vahvistaa oppilaan kykyä laskea peruslaskutoimituksia rationaaliluvuilla	S11	L1, L4
T12	laajentaa lukukäsitteen ymmärtämistä reaalilukuihin	S11	L1, L4
T13	laajentaa oppilaan ymmärrystä prosenttilaskennasta	S11, S15	L1, L3, L6
T14	ohjaa oppilasta ymmärtämään tuntemattoman käsitteen ja kehittää oppilaan yhtälönratkaisutaitoja	S12, S13	L1, L4
T15	ohjaa oppilasta ymmärtämään muuttujan käsitteen ja tutustuttaa funktion käsitteeseen. Harjoittaa funktion kuvaajan tulkitsemista ja tuottamista.	S12, S13	L1, L4, L5
T16	laajentaa oppilaan ymmärrystä geometrian käsitteistä ja niiden välisistä yhteyksistä.	S14	L1, L4, L5
T17	ohjaa oppilasta ymmärtämään ja hyödyntämään suorakulmaiseen kolmioon ja ympyrään liittyviä ominaisuuksia.	S14	L1, L4, L5
T18	vahvistaa oppilaan taitoa laskea pinta- aloja ja tilavuuksia.	S14	L1, L4
T19	ohjaa oppilasta määrittämään tilastollisia tunnuslukuja ja laskemaan todennäköisyyksiä	S15	L3, L4, L5
T20	harjoittaa oppilaan algoritmista ajattelua sekä taitoa soveltaa matematiikkaa ja ohjelmointia ongelmien ratkaisemiseen	S10	L1, L4, L5, L6

Tarkemmat arvosteluperiaatteet

	Arvioinnin kohteena oppiaineessa	< 8 osaaminen	Arvosanan kahdeksan osaaminen	> 8 osaaminen
T20	Algoritminen ajattelu ja ohjelmointitaito	Oppilas ymmärtää algoritmisen ajattelun periaatteita ja ohjelmoinnin peruskäsitteitä.	Oppilas ymmärtää algoritmisen ajattelun periaatteita ja osaa ohjelmoida yksinkertaisia ohjelmia	Oppilas osaa suunnitella ja tuottaa ohjelmia

Keskeiset sisältöalueet

S10 Ajattelun taidot ja menetelmät

1. Harjoitellaan loogista ajattelua vaativia toimintoja kuten sääntöjen ja riippuvuuksien etsimistä ja esittämistä täsmällisesti.
2. Opitaan määrittämään vaihtoehtojen lukumääriä.
3. Vahvistetaan oppilaan päättelykykyä ja taitoa perustella.
4. Harjoitellaan matemaattisen tekstin tulkitsemista ja tuottamista.
5. Tutustutaan todistamisen perusteisiin.
6. Harjoitellaan väitelauseiden totuusarvon päättelyä.
7. Syvennetään algoritmista ajattelua.
8. Ohjelmoidaan ja samalla harjoitellaan hyviä ohjelmointikäytäntöjä.
9. Sovelletaan itse tehtyjä ja valmiita tietokoneohjelmia osana matematiikan opiskelua.

S11 Luvut ja laskutoimitukset

1. Harjoitellaan peruslaskutoimituksia myös negatiivisilla luvuilla.
2. Vahvistetaan laskutaitoa murtoluvuilla ja opitaan murtoluvun kertominen ja jakaminen murtoluvulla.
3. Tutustutaan vastaluvun, käänteisluvun ja itseisarvon käsitteisiin.
4. Lukualuetta laajennetaan reaalilukuihin.
5. Perehdytään lukujen jaollisuuteen ja opitaan jakamaan luku alkutekijöihin.
6. Syvennetään desimaalilukujen laskutoimituksien osaamista.
7. Vahvistetaan ymmärrystä tarkan arvon ja likiarvon erosta sekä pyöristämisestä.
8. Varmistetaan prosentin käsitteen ymmärrys.
9. Harjoitellaan prosenttiosuuden laskemista ja prosenttiluvun osoittama määrän laskemista kokonaisuudesta. Lisäksi opitaan laskemaan muuttunut arvo, perusarvo sekä muutos- ja vertailuprosentti.
10. Harjoitellaan potenssilaskentaa, kun eksponenttina on kokonaisluku.
11. Opitaan neliöjuuren käsite ja käytetään neliöjuurta laskutoimituksissa.

S12 Algebra

1. Perehdytään muuttujan käsitteeseen ja lausekkeen arvon laskemiseen.
2. Harjoitellaan potenssilausekkeiden sieventämistä.
3. Tutustutaan polynomin käsitteeseen ja harjoitellaan polynomien yhteen-, vähennys- ja kertolaskua.
4. Harjoitellaan muodostamaan lausekkeita ja sieventämään niitä.
5. Muodostetaan ja ratkaistaan ensimmäisen asteen yhtälöitä ja vaillinaisia toisen asteen yhtälöitä.
6. Ratkaistaan yhtälöpareja graafisesti ja algebrallisesti.
7. Tutustutaan ensimmäisen asteen epäyhtälöihin ja ratkaistaan niitä.
8. Syvennetään oppilaan taitoa tutkia ja muodostaa lukujonoja.
9. Käytetään verrantoa tehtävien ratkaisussa.

S13 Funktiot

1. Opitaan kuvaamaan riippuvuuksia sekä graafisesti että algebrallisesti.
2. Tutustutaan suoraan ja kääntäen verrannollisuuteen.
3. Perehdytään funktion käsitteeseen.
4. Piirretään suoria ja paraabeleja koordinaatistoon.
5. Opitaan suoran kulmakertoimen ja vakiotermin käsitteet.
6. Tulkitaan kuvaajia esimerkiksi tutkimalla funktion kasvamista ja vähenemistä.
7. Opitaan määrittämään funktioiden nollakohtia.

S14 Geometria

1. Laajennetaan pisteen, viivan, janan, puolisuoran, suoran ja kulman käsitteiden ymmärtämistä.
2. Tutkitaan suoriin, kulmiin ja monikulmioihin liittyviä ominaisuuksia.
3. Vahvistetaan yhdenmuotoisuuden ja yhtenevyyden käsitteiden ymmärtämistä.
4. Harjoitellaan geometrista konstruointia.
5. Opitaan käyttämään Pythagoraan lausetta, Pythagoraan käänteislausetta ja trigonometrisia funktioita.
6. Opitaan kehä- ja keskuskulma sekä tutustutaan Thaleen lauseeseen.
7. Lasketaan monikulmioiden piirejä ja pinta-aloja.
8. Opitaan laskemaan ympyrän kehän pituus ja pinta-ala. Lisäksi opitaan laskemaan ympyrän kaaren pituus ja sektorin pinta-ala.
9. Tutkitaan kolmiulotteisia kappaleita. Opitaan laskemaan pallon, lieriön ja kartion pinta-ala ja tilavuus.
10. Varmennetaan ja laajennetaan mittayksiköiden ja yksikkömuunnosten hallintaa.

S15 Tietojen käsittely ja tilastot sekä todennäköisyys

1. Syvennetään oppilaan taitoja kerätä, jäsentää ja analysoida tietoa.
2. Varmistetaan keskiarvon ja tyyppiarvon ymmärtäminen.
3. Opitaan määrittämään frekvenssi, suhteellinen frekvenssi ja mediaani.
4. Tutustutaan hajonnan käsitteeseen.
5. Opitaan tulkitsemaan ja tuottamaan erilaisia diagrammeja.
6. Opitaan laskemaan todennäköisyyksiä.

LUKU 3

Keskeneräiset aiheet

3.1 Sanasto

argument, argumentti, parameter, parametri Funktiokutsussa argumentteja ovat kutsulle funktiolle annettavat tiedot, esimerkiksi `(funktio 1 2)`. Parametrit taas toimivat niminä näille argumenteille funktion määrittelyn sisällä. Esimerkiksi `(define (funktio parametri1 parametri2) (+ parametri1 parametri2)` laskisi ylläolevan esimerkin tapauksessa summan luvuista 1 ja 2.

function call, funktiokutsu TODO

recursion, rekursio TODO

tail-recursion, häntärekursio TODO

function, funktio TODO

boolean TODO

float, liukuluku TODO

bit, bitti TODO

byte, tavu TODO

binary, binääri TODO

hexdecimal, heksadesimaali TODO

list, lista TODO

(singly) linked list, linkitetty lista TODO

array, taulukko TODO

stack, pino TODO

heap, keko TODO

tree, puu TODO

graph, graafi TODO

vertex, solmu TODO

edge, kaari TODO

value, arvo TODO

number, numero TODO

character, merkki TODO

string, merkkijono TODO

string/text encoding, merkistökoodaus TODO

library, kirjasto TODO

constant, vakio TODO

variable, muuttuja TODO

scope, näkyvyysalue TODO

global, yleinen (globaali, ohjelman laajuinen, yleis-?) TODO

local, paikallinen TODO

closure, sulkeuma TODO

struct, tietue TODO

hash table/array, hajautustaulu TODO

hash function, hajautusfunktio TODO

tuple, monikko TODO

macro, makro TODO

interpreter, tulkki TODO

compiler, kääntäjä TODO

type, tyyppi TODO

definition, määritelmä TODO

3.2 Mitä tehdä, jos DrRacket antaa virheilmoituksia

- *X: this function is not defined*
- *define: expected only one expression after the variable name X, but found 1 extra part*
- *function call: expected a function after the open parenthesis, but found a part*
- *define: expected only one expression for the function body, but found 1 extra part*

3.2.1 X: this function is not defined

Virheen ensimmäinen sana riippuu siitä, minkä nimistä funktiota olet yrittänyt kutsua. Virhe voi johtua kahdesta syystä: joko olet kirjoittanut funktion nimen väärin, tai olet unohtanut käyttää `require`-funktiota ohjelman alussa. Esimerkiksi DrRacket ei löydä piirtokomentoja ennenkuin ohjelmassa on `(require 2htdp/image)`.

Jos kutsut omaa funktiotasi, tarkista myös että olet kirjoittanut funktion nimen oikein määritelmässä! DrRacket ei ymmärrä mitkä sanat ovat oikeaa suomea (tai englantia), vaan tarkastaa ainoastaan että määritelmässä annettu ja kutsussa käytetty nimi täsmäävät.

3.2.2 define: expected only one expression after the variable name X, but found 1 extra part

Tämä virhe toi tulla määritellessä funktioita, jos unohdit laittaa sulut funktion ja parametrien nimien ympärille. Racket luulee, että yrität nimetä vakiota, mutta hämääntyy kun nimen jälkeen tuleekin monta asiaa.

Esimerkiksi

```
(define lisääjä luku
  (+ luku 1))
```

aiheuttaa tämän virheilmoituksen. DrRacket myös maalaa punaisella ylimääräiset osat, yhden parametrin tapauksessa funktion sisällön. Oikein kirjoitettu funktio olisi

```
(define (lisääjä luku)
  (+ luku 1))
```

3.2.3 function call: expected a function after the open parenthesis, but found a part

Tässä quote:*open parenthesis* ei tarkoita ylimääräistä tai sulkematonta sulkua, vaan yleensä aloittavaa sulkua. Racketissä aloittavan sulun jälkeen pitää tulla aina funktion nimi. Esimerikssi funktiokutsussa

```
((circle 10 "solid" "red"))
```

Ongelmana on kutsun ympärillä olevat ylimääräiset sulut. Oikein kirjoitettu kutsu on

```
((circle 10 "solid" "red"))
```

Monimutkaisemmassa esimerkissä

```
(overlay/offset
 (overlay/offset ((circle 10 "solid" "red") 10 10 (circle 10 "solid" "red"))
                 20 20
                 (rectangle 20 40 "solid" "blue")))
```

Ongelmana on väärään paikkaan lipsahtanut alkava sulku, ja aivan ohjelman loppuun tullut ylimääräinen sulku. Jos annat Racketin sisentää koodisi automaattisesti, kuten esimerkissä, kolmas ja neljäs rivi sisentyvät samaan kohtaan kuin ensimmäinen circle-kutsu. Tämä tarkoittaa, että DrRacket pitää näitä kaikkia sisemmän overlay/offsetin argumentteina.

Asian voi myös varmistaa viemällä kursori viimeisen rivin loppuun, jolloin DrRacket maalaa koko koodin harmaalla. Tämä tarkoittaa, että aivan ensimmäinen alkava ja viimeinen sulkeva sulku vastaavat toisiaan (eli kaikille muille alkaville suluille on löytynyt pariksi aiempi sulkeva sulku). Jos kursoria vie nyt yhden kirjaimen vasemmalle, toiseksi viimeiseen sulkevaan sulkuun, DrRacket maalaa koko sisemmän kutsun. Eli ulommalla overlay/offsetillä on liian vähän argumentteja, ja sisemmällä liian paljon.

Itse virheilmoitus tulee ensimmäistä circleä edeltävästä tuplasulusta, joista toinen pitää poistaa. Samoin aivan viimeinen sulkeva sulku pitää poistaa, jotta sulkuja olisi oikea määrä. Jos nyt maalaat hiirellä koko koodin ja painat `tab`-näppäintä (`Q`-näppäimen vasemmalla puolen), DrRacket sisentää koodin uudelleen:

```
(overlay/offset
 (overlay/offset (circle 10 "solid" "red") 10 10 (circle 10 "solid" "red"))
 20 20
 (rectangle 20 40 "solid" "blue"))
```

Nyt kolmas ja neljäs rivi rivittyvät sisemmän overlay/offsetin tasolle, eli ne kaikki ovat argumentteja ulommalle overlay/offsetille.

3.2.4 define: expected only one expression for the function body, but found 1 extra part

Funktiomääritelmässäsi on liikaa osia, esimerkiksi:

```
(define (lisääjä x)
  (+ x 1)
  (- x 1))
```

Funktio yrittää siis tehdä kahta asiaa, sekä lisätä että vähentää annetusta parametrista yhden. Virheen korjataksesi sinun täytyy poistaa jompikumpi, tai ehkä tarkoituksesi oli antaa nämä vielä jollekin kolmannelle funktiolle argumenteiksi.

3.3 Funktioiden suunnittelu ja testaus

3.3.1 1. Mieti ja kirjoita ylös missä muodossa tieto on

Esimerkiksi lämpötilat, etäisyydet ja kappaleen dimensiot ovat desimaalinumeroita; nimet ja osoitteet tekstiä; ja kuvat kuvia.

3.3.2 2. Kirjoita funktion syötteet ja ulostulot, tarkoitus ja funktion yläosa

Syötteet tulevat välein eroteltuna ensin, ja sitten nuolen jälkeen ulostulot. Käytä Racketin tietotyyppejä kuvaamaan molempia.

Sen jälkeen kirjoita selväkielinen, lyhyt kuvaus funktion tarkoituksesta.

Lopuksi tee funktion kuvaus, joka sisältää funktion ja parametrien nimet. Funktio voi tässä vaiheessa palauttaa minkä tahansa ulostulon tyypin arvon.

Esimerkiksi

```
; Number -> Number
; muuta asteet Celsiusta Kelvineiksi
(define (celsius->Kelvin C)
    0)
```

```
; Number -> Image
; Piirrä pallo, jonka säde annetaan
(define (piirrä-pallo säde)
    (empty-scene))
```

3.3.3 3. Mieti esimerkkejä syötteestä ja oikeasta ulostulosta, ja kirjoita ne testeiksi

```
; Number -> Number
; muuta asteet Celsiusta Kelvineiksi
(check-expect (celsius-Kelvin 0) 273.15)
(check-expect (celsius-Kelvin -273.15) 0)
(define (celsius->Kelvin C)
    0)
```

3.3.4 4. Hahmottele funktiota

Mieti mitä kaikkea funktio ainakin tarvitsee toimiakseen. Muodosta hahmotelma, johon ne on sijoitettu. Hahmotelmaan tulee vähintään funktion parametrit.

```
; Number -> Number
; muuta asteet Celsiusta Kelvineiksi
(check-expect (celsius-Kelvin 0) 273.15)
(check-expect (celsius-Kelvin -273.15) 0)
(define (celsius->Kelvin C)
    (... C ...))
```

Hahmotelma ei ole oikeaoppisesti ohjelma, joten tässä vaiheessa ohjelman suorittaminen valittaisi ongelmista. Sen tarkoitus on auttaa miettimään ohjelman rakennetta.

3.3.5 5. Koodaa!

```
; Number -> Number
; muuta asteet Celsiusta Fahrenheiteiksi
(check-expect (celsius->fahrenheit 0) 32)
(check-expect (celsius->fahrenheit 100) 212)
(check-expect (celsius->fahrenheit -40) -40)
(define (celsius->fahrenheit C)
    (+ 32 (* 9/5 C)))
```

3.3.6 6. Testaa

Kun käytät `check-expect` funktiota, voit helposti testata funktion toimivuuden suorittamalla ohjelman. Jos ohjelma toimii, olet valmis. Jos ei, ohjelmassa voi olla virhe ja sinun täytyy palata edelliseen kohtaan. On myös mahdollista, että testeissä on virhe, joten jos et löydä koodista ongelmia, varmista että testisi ovat oikein.

Tehtävä vielä

Ostoslista, top-down

3.4 Animaatiot

Yksinkertainen animaatio yhteen suuntaan yhdellä kappaleella (maailmaksi riittää yksinkertainen datatyyppi).

```
(big-bang maailma
          (on-tick tick-expr)
          (to-draw draw-expr))
```

3.5 Tietueet

Tehtävä vielä

Rakenteet? Miten tämä käännetään fiksusti...

Tehtävä vielä

Ensin vaikka make-posn, sitten tehdään oma 3D:ssä

3.6 Interaktiivisuus animaatioissa

```
(big-bang maailma
          (on-tick tick-expr)
          (on-key key-expr)
          (to-draw draw-expr))
```

3.7 Listat ja rekursio

Fig. 3.1: A singly linked list of three numbers.

Tehtävä vielä

Esimerkki listan summasta ja kertomasta: Tyhjän listan tapaus eri! Listan neliöiden summa. Listan konkatenointi. Listan koon laskeminen. Filteröinti (eri tehtävissä erilailla?). Listan muuttaminen kuvaksi.

```
(require 2htdp/image)

(check-expect (summa (list 1 2 3 4)) 10)
(define (summa lista)
  (cond [(empty? lista) 0]
        [else (+ (first lista) (summa (rest lista)))]))

(check-expect (kertoma (list 1 2 3 4)) 24)
(define (kertoma lista)
  (cond [(empty? lista) 1]
        [else (* (first lista) (kertoma (rest lista)))]))

(check-expect (neliöt (list 1 2 3 4)) 30)
(define (neliöt lista)
  (cond [(empty? lista) 0]
        [else (+ (expt (first lista) 2) (neliöt (rest lista)))]))

(check-expect (konkatenoi (list "Moi" "," " " "maailma"))
              "Moi, maailma")
(define (konkatenoi lista)
  (cond [(empty? lista) ""]
        [else (string-append (first lista) (konkatenoi (rest lista)))]))

(check-expect (koko (list 1 2 3 4)) 4)
(define (koko lista)
  (cond [(empty? lista) 0]
        [else (+ 1 (koko (rest lista)))]))

(check-expect (yli-kolme (list 1 2 3 4 5)) (list 4 5))
```

```
(define (yli-kolme lista)
  (cond [(empty? lista) empty]
        [(> (first lista) 3) (cons (first lista)
                                   (yli-kolme (rest lista)))]
        [else (yli-kolme (rest lista))]))

(check-expect (helmet (list 1 2))
              (beside (circle 1 "solid" "blue")
                      (circle 2 "solid" "blue")))
(define (helmet lista)
  (cond [(empty? lista) empty-image]
        [else (beside (circle (first lista) "solid" "blue")
                      (helmet (rest lista)))]))
```

3.7.1 Kiinnostuitko?

Tehtävä vielä

Puut onkin nyt oma kappaleensa... tarvitsee muuta

Fig. 3.2: A binary tree.

3.8 Algoritmit

3.9 Lamda ja funktiot parametreina

3.10 Puut ja graafit

```
raco pkg install graph
```

BSL with list abbreviations

Pitää olla quote selitettynä

Solmu, kaari, graafi, sykli, syklitön (puu)

Suunnattu ja suuntaamaton

Painotettu ja painottamaton

```
(require graph)
(define g (weighted-graph/undirected
           '((178 Helsinki Tampere)
             (168 Helsinki Turku)
             (162 Turku Tampere)
             (104 Helsinki Lahti)
             (63 Kouvola Lahti)
             (127 Lahti Tampere)
             (487 Oulu Tampere)
             (287 Oulu Kuopio)
             (320 Oulu Vaasa)
```

```
              (242 Vaasa Tampere)
              (335 Vaasa Turku)
              (290 Lahti Kuopio)
              (292 Kuopio Tampere)
              (377 Kuopio Vaasa)
              (267 Kuopio Kouvola))))
(get-vertices g)
(get-edges g)
(edge-weight g 'Helsinki 'Lahti)
```

3.10.1 Lyhyimmän matkan etsiminen kahden kaupungin välillä

Leveyshaku

Syvyyshaku

Djikstra

3.10.2 Kauppamatkustajan ongelma

3.11 Funktion sisäiset määritelmät

`Intermediate Student` ja local.

3.12 Muuttujat ja silmukat

Muuttujat vaativat ASL:n.

Silmukat vaativat racket/base'in.

3.13 Sivuvaikutukset

`begin` vaatii ASL:n.

3.14 Algoritmien laskennallinen vaativuus

O-notaatio, P, NP

3.15 Tiedostojen luku ja kirjoittaminen

3.16 Tiedostojen käsittely listoina

3.17 Tietotyypit

Konetta lähempänä olevat tyypit: int8, int16, int32, int64 eli short, int, long, long long.

Samat flotareille.

Puhetta vahvasti tyypitetyistä kielistä ja ohjelmointikielen tyyppien erosta matalan tason tyyppeihin.

3.18 Liukuluvut

Rikotaan jakolaskuilla/neliöjuurilla ja palautuksella tarkkuus.

3.19 Tietokoneen rakenne

Kaikissa suoritin, rekistereitä, muistia, muita laitteita

Puhetta nykyisistä x86/amd64 laitteista: monta ydintä, RAM, HDD/SDD, näytönohjain, verkkokortti, äänikortti.

Ehkä jotain integroinnista CPU:n.

Mahdollisesti verrata Arduinoon ja Raspiin.

3.20 Tietoverkot

IPv4, DNS, DHCP, SSL/TLS, SMTP, HTTP

IP ei välttämättä ole sama kone, vaan se riippuu siitä minne liikenne ohjataan. Esimerkiksi Netflixillä saattaa olla kone operaattorin verkossa.

Kansainväliset tietoverkot, ja miksi kaivuri voi katkaista Elisan, ja miten jotkut sivut toimivat mutta toiset eivät samaan aikaan.

DDOS, tahallinen ja tahaton (slashdotting).

3.20.1 Mitä tapahtuu kun pyydät nettiselainta avaamaan sivun?

```
> route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         OpenWrt.lan     0.0.0.0         UG    0      0        0 eth1
```

> dig f.root-servers.net +showsearch +trace ;; Received 913 bytes from 127.0.1.1#53(127.0.1.1) in 255 ms ;; Received 734 bytes from 192.5.5.241#53(f.root-servers.net) in 71 ms ;; Received 660 bytes from 192.43.172.30#53(i.gtld-servers.net) in 51 ms ;; Received 156 bytes from 216.239.36.10#53(ns3.google.com) in 20 ms Googlen IP osoite on siis esimerkiksi 80.239.229.250.

```
> traceroute 80.239.229.250
traceroute to 80.239.229.250 (80.239.229.250), 30 hops max, 60 byte packets
 1  OpenWrt.lan (192.168.1.1)  1.434 ms  1.572 ms  1.906 ms
 2  dsl-espbrasgw1-54f9c0-1.dhcp.inet.fi (84.249.192.1)  2.226 ms  2.352 ms  2.932 ms
 3  hls-b2-link.telia.net (62.115.143.50)  3.830 ms  4.287 ms hls-b1-link.telia.net (62.115.143.84)
 4  google-ic-154333-hls-b2.c.telia.net (80.239.195.206)  4.855 ms hls-b2-link.telia.net (80.91.252.1
 5  google-ic-154333-hls-b2.c.telia.net (80.239.195.206)  5.845 ms * *
 6  * * *
```

```
> nc 80.239.229.250 80
GET / HTTP/1.1
Host: google.com
```

```
HTTP/1.1 302 Found
Cache-Control: private
Content-Type: text/html; charset=UTF-8
Location: http://www.google.fi/?gfe_rd=cr&ei=k9qJVKuLA8HO-ga264CYCA
Content-Length: 258
Date: Thu, 11 Dec 2014 17:55:31 GMT
Server: GFE/2.0
Alternate-Protocol: 80:quic,p=0.002

<HTML><HEAD><meta http-equiv="content-type" content="text/html;charset=utf-8">
<TITLE>302 Moved</TITLE></HEAD><BODY>
<H1>302 Moved</H1>
The document has moved
<A HREF="http://www.google.fi/?gfe_rd=cr&amp;ei=k9qJVKuLA8HO-ga264CYCA">here</A>.
</BODY></HTML>
```

www.google.com on 173.194.40.255

```
> traceroute 173.194.40.255
traceroute to 173.194.40.255 (173.194.40.255), 30 hops max, 60 byte packets
 1  OpenWrt.lan (192.168.1.1)  7.798 ms  14.358 ms  19.065 ms
 2  dsl-espbrasgw1-54f9c0-1.dhcp.inet.fi (84.249.192.1)  23.126 ms  27.840 ms  37.507 ms
 3  hls-b1-link.telia.net (62.115.143.84)  38.288 ms hls-b2-link.telia.net (62.115.143.50)  38.073 ms
 4  s-bb3-link.telia.net (80.91.245.136)  51.148 ms s-bb4-link.telia.net (213.155.133.78)  51.239 ms
 5  s-b5-link.telia.net (213.155.133.19)  58.176 ms s-b5-link.telia.net (62.115.141.203)  80.825 ms s
 6  google-ic-306509-s-b5.c.telia.net (62.115.45.14)  88.733 ms  13.966 ms  13.204 ms
 7  209.85.250.192 (209.85.250.192)  13.623 ms  15.627 ms  16.861 ms
 8  209.85.253.151 (209.85.253.151)  19.255 ms  19.378 ms  16.971 ms
 9  arn02s06-in-f31.1e100.net (173.194.40.255)  20.003 ms  19.852 ms  20.101 ms
```

```
> nc 173.194.40.255 80
GET /?gfe_rd=cr&amp;ei=k9qJVKuLA8HO-ga264CYCA HTTP/1.1
Host: www.google.fi
```

paljon dataa.

3.21 Piirretään sektoreita

```
(require 2htdp/image)
(define SIZE 50)
(define (make-wedge color angle)
   (underlay/align
    "left" "top"
```

```
    (circle SIZE "solid" (make-color 255 255 255 0))
    (crop 0 0 (* 2 SIZE) SIZE
          (cond
            [(< angle 90)
             (underlay/align "left" "bottom"
                             (circle SIZE "solid" (make-color 255 255 255 0))
                             (rotate (- 180 angle)
                                     (crop 0 0 (* 2 SIZE) SIZE
                                           (circle SIZE "solid" color))))]
            [else
             (underlay/align "left" "top"
                             (circle SIZE "solid" (make-color 255 255 255 0))
                             (rotate (- 180 angle)
                                     (crop 0 0 (* 2 SIZE) SIZE
                                           (circle SIZE "solid" color))))])))

(overlay
 (rotate 240 (make-wedge "red" 120))
 (rotate 120 (make-wedge "blue" 120))
 (make-wedge "green" 120))
```

```
(overlay
 (rotate 300 (make-wedge "red" 60))
 (rotate 240 (make-wedge "violet" 60))
 (rotate 180 (make-wedge "blue" 60))
 (rotate 120 (make-wedge "green" 60))
 (rotate 60 (make-wedge "yellow" 60))
 (make-wedge "orange" 120))
```

HSL.

```
(define (hue-to-color hue)
  (f1 (/ hue 60)))

(define (my-modulo a b)
  (cond
    [(< a b) a]
    [else (my-modulo (- a b) b)]))
```

```
(define (f1 H2)
  (f2 H2 (round (* 255 (- 1 (abs (- (my-modulo H2 2) 1)))))))

(define (f2 H2 X)
  (cond
    [(<= 0 H2 1) (make-color 255 X 0)]
    [(<= 1 H2 2) (make-color X 255 0)]
    [(<= 2 H2 3) (make-color 0 255 X)]
    [(<= 3 H2 4) (make-color 0 X 255)]
    [(<= 4 H2 5) (make-color X 0 255)]
    [(<= 5 H2 6) (make-color 255 0 X)]))

(define (lots-of-wedges number)
  (make-more-wedges number 0))

(define (make-more-wedges how-many angle)
  (cond
    [(>= angle 360) (make-wedge "white" 0)]
    [else
     (overlay
      (rotate angle (make-wedge (hue-to-color angle) (/ 360 how-many)))
      (make-more-wedges how-many (+ angle (/ 360 how-many))))]))

(lots-of-wedges 60)
```

Hieman toistettua koodia vähentämällä:

```
(require 2htdp/image)
(define SIZE 50)
(define alpha-circle
  (circle SIZE "solid" (make-color 255 255 255 0)))

(define (half image)
  (crop 0 0 (* 2 SIZE) SIZE image))

(define (make-wedge color angle)
   (underlay/align
    "left" "top"
    alpha-circle
    (half (underlay/align
           "left"
           (cond [(< angle 90) "bottom"]
                 [else "top"])
           alpha-circle
           (rotate (- 180 angle)
                   (half (circle SIZE "solid" color)))))))

(define (hue-to-color hue)
  (f1 (/ hue 60)))
```

```
(define (f1 H2)
  (f2 H2 (round (* 255 (- 1 (abs (- (my-modulo H2 2) 1)))))))

(define (my-modulo a b)
  (cond [(< a b) a]
        [else (my-modulo (- a b) b)]))

(define (f2 H2 X)
  (cond [(<= 0 H2 1) (make-color 255 X 0)]
        [(<= 1 H2 2) (make-color X 255 0)]
        [(<= 2 H2 3) (make-color 0 255 X)]
        [(<= 3 H2 4) (make-color 0 X 255)]
        [(<= 4 H2 5) (make-color X 0 255)]
        [(<= 5 H2 6) (make-color 255 0 X)]))
```

3.22 Lisää animaatioista

Animaatiot structilla, esimerkiksi 2D-pelit.

3.23 Monen kappaleen animointi

3.24 Turtle-grafiikka

```
(require graphics/value-turtles)

(define sivun-pituus 40)
(define ruudun-koko 150)

(define (kulma n asteet turtle)
    (cond
          [(= n 0) turtle]
              [else (kulma (- n 1) asteet (turn asteet (draw sivun-pituus turtle)))]))

(define (monikulmio n)
    (kulma n (/ 360 n) (turtles ruudun-koko ruudun-koko)))

(monikulmio 3)
(monikulmio 4)
(monikulmio 5)
(monikulmio 6)
```

3.25 Musiikkia

```
raco pkg install rsound
```

```
(require rsound)
(require rsound/piano-tones)

(define (piano-list number-list time)
  (cond [(empty? number-list) empty]
```

```
        [else (cons (list (piano-tone (first number-list))
                          (* 22050 time))
                    (piano-list (rest number-list)
                                (+ time 1)))]))

(play
 (assemble
  (piano-list
   (list 60 60 60 64 62 62 62 65 64 64 62 62 60) 0)))
```

3.26 Salasanojen vahvuus

```
(require (planet mbenkard/mulkrypt:1:2/cubehash))

(cubehash-128 #"lyhyt")
(cubehash-128 #"salasana")
(cubehash-128 #"pitkäsalasana")
```

Muista: planet-funktio hakee paketin Racketin käyttäjien tekemien kirjastojen joukosta.

3.26.1 Tavut

On erilaisia lukujärjestelmiä. Racketissä voidaan kirjoittaa numeroita binäärinä, eli kaksikantaisena, ja heksadesimaalina, eli 16-kantaisena laittamalla lukujen eteen liitteet `#b` ja `#x`. Heksadesimaalissa yhdeksää suuremmat numerot esitetään kirjaimilla a:sta f:n.

Tietokoneet tallentavat kaiken tietonsa sisäisesti binäärilukuina, jonka yksittäisiä osia sanotaan biteiksi. Yleensä muisti on jaoteltu yhden tai useamman *tavun*, eli kahdeksan bitin osiin.

Yksi tavu voi siis sisältää lukuja väliltä `#b00000000`-`#b11111111`, eli 0-255. Heksadesimaalina tavut vaativat vain kaksi merkkiä. Heksat sisältävät siis neljän bitin verran tietoa.

3.26.2 Merkistökoodaukset

Koska kaikki tietokoneen sisällä oleva tieto on tavuihin jaettuja bittejä, myös teksti täytyy esittää jollain lailla binäärinumeroina. Sopimusta mikä numero vastaa mitäkin kirjainta sanotaan *merkistökoodaukseksi*.

Vanhin tietokoneissa käytetty merkistökoodaus on ASCII. Siinä jokaista kirjainta, kuten myös erikoismerkkiä, vastasi seitsemän bittiä. Siihen mahtuu englanninkieliseen tekstiin riittävä määrä kirjaimia, mutta ei ääkkösiä. Muita kieliä varten käyttöön otettiin kahdeksas bitti, mutta yli 127:n menevät numerot tarkoittivat eri kielissä eri kirjaimia.

3.26.3 Hajautusalgoritmi

eli hash-funktio.

Muuttaa minkä tahansa määrän tavuja ennaltamäärätyksi määräksi tavuja.

Pieni muutos syötteessä muuttaa ulostuloa paljon.

Hyvin hankala laskea toiseen suuntaan, eli mikä on ollut syöte jos tiedetään ulostulo.

Käytetään salasanojen tallentamiseen, sillä jos salasanatiedosto varastetaan, varkaan on hyvin hankala selvittää käytettyjä salasanoja ja kokeilla samoja myös muihin paikkoihin.

Suurin mahdollinen 128-bittinen numero on binäärinä 128 ykköstä, eli heksadesimaalina 128 : 4 f:ä: `#xffffffffffffffffffffffffffffffff`. Desimaalijärjestelmässä luku on 340282366920938463463374607431768211455.

3.26.4 Brute force

Kokeillaan yksi kerrallaan kaikkia mahdollisia salasanoja.

3.26.5 Sanasto

Kokeillaan valmiissa sanastossa olevia, yleisesti käytettyjä sanoja.

3.26.6 Sanasto muutoksilla

Kokeillaan yhdistää sanoja, ja muuttaa niitä tietyillä yleisillä säännöillä, kuten vaihtamalla A-kirjaimet numeroksi 4, tai E 3:ksi.

3.27 Tekijöihin jako ja alkuluvut

```
(define (tekijä luku jakaja)
  (cond
    [(= luku 1)
     empty]
    [(= (modulo luku jakaja) 0)
     (cons jakaja (tekijä (/ luku jakaja) 2))]
    [else
     (tekijä luku (+ jakaja 1))]))

(check-expect (etsi-tekijät 10) (list 2 5))
(check-expect (etsi-tekijät 11) (list 11))
(check-expect (etsi-tekijät 300) (list 2 2 3 5 5))
(define (etsi-tekijät luku)
  (tekijä luku 2))

(check-expect (alkuluku? 2) #true)
(check-expect (alkuluku? 3) #true)
(check-expect (alkuluku? 11) #true)
(check-expect (alkuluku? 4) #false)
(check-expect (alkuluku? 15) #false)
(define (alkuluku? luku)
    (= (length (etsi-tekijät luku)) 1))

(check-expect (etsi-alkuluvut 0 100) empty)
(check-expect (etsi-alkuluvut 1 2) (list 2))
(check-expect (etsi-alkuluvut 7 2) (list 2 3 5 7 11 13 17))
(check-expect (etsi-alkuluvut 1 15) (list 17))
(define (etsi-alkuluvut kuinka-monta aloitus)
  (cond
    [(= kuinka-monta 0)
```

```
     empty]
    [(alkuluku? aloitus)
     (cons aloitus (etsi-alkuluvut (- kuinka-monta 1) (+ aloitus 1)))]
    [else
     (etsi-alkuluvut kuinka-monta (+ aloitus 1))]))
```

Tehtävä vielä

Kommentoitu versio ja vertailu helppolukuisuudesta

Tehtävä vielä

Nopeampi versio, joka tarkistaa vain neliöjuureen asti, ja kolmas joka tarkistaa jakamalla aikaisemmilla alkuluvuilla

Tehtävä vielä

Katsoa wikipediasta oikeasti nopea algoritmi

Tehtävä vielä

Vertailu kertolaskun ja tekijöiden etsimisen nopeudesta: RSA

3.28 Tiedon hakeminen suoraan Internetistä

```
(require racket)
(require net/url)
(require json)

(define JSON
  "{\"coord\":{\"lon\":25.47,\"lat\":65.01},\"sys\":{\"type\":1,\"id\":5036,\"message\":0.1024,\"cou

(hash-ref
  (hash-ref
    (string->jsexpr JSON)
    'main)
  'temp)

(hash-ref
  (hash-ref
    (string->jsexpr
      (bytes->string/utf-8
        (port->bytes
          (get-pure-port
            (string->url "http://api.openweathermap.org/data/2.5/weather?q=Oulu,fi")))))
    'main)
  'temp)
```

3.29 Funktioiden piirtäminen kuvaajiksi

Vaatii `Intermediate Student`-kielen.

Vaatii Ehtolauseet ja boolean-logiikka, Listat ja rekursio, Funktion sisäiset määritelmät.

Tehtävä vielä

Ensin animoimaton versio

Tehtävä vielä

Sopivan step-sizen etsintä

```
(require 2htdp/universe)
(require 2htdp/image)

(define WIDTH 600)
(define HEIGHT 600)
(define MAX_X 5)
(define MAX_Y 1.4)


(define (draw world)
    (local ((define (f x) (sin (+ x world))))
           (plot f (- MAX_X) 0.2 (empty-scene WIDTH HEIGHT))))

(define (plot function start step scene)
    (cond
        [(> (+ step start) MAX_X) scene]
        [else (plot function (+ start step) step
                    (add-line
                        scene
                        (scale-x start)
                        (scale-y (function start))
                        (scale-x (+ step start))
                        (scale-y (function (+ step start)))
                        "black"))]))

(define (scale-x value)
    (+ (* (/ value 2 MAX_X) WIDTH)
       (/ WIDTH 2)))

(define (scale-y value)
    (+ (* -1 (/ value 2 MAX_Y) HEIGHT)
       (/ HEIGHT 2)))

(define (step world)
    (+ world 0.1))

(big-bang 0
    [to-draw draw]
    [on-tick step])
```

Tehtävä

Miten tekisit seuraavat funktiot? Huomaa, että joudut säätämään y:n maksimirajaa, jotta kuva piirtyy oikein.

Tehtävä vielä

Antaako valmiina vastauksena:

```
(λ (x) (* 2 (+ x world)))
(λ (x) (expt (+ x world) 2))
(λ (x) (sin (expt x 4)))
(λ (x) (sin (exp x)))
(λ (x) (tan x))
(λ (x) (sin (/ x)))
```

Pitäisikö varottaa, että tan räjähtää äärettömään ja viimeisestä tulee jako nollalla, vaiko jättää oppimiskokemukseksi?

3.29.1 Monen funktion plottaus

```
(define (draw world)
    (local ((define (f x) (sin (+ x world)))
            (define (g x) (cos (+ x world))))
           (plot g (- MAX_X) 0.1
                 (plot f (- MAX_X) 0.2 (empty-scene WIDTH HEIGHT)))))
```

3.29.2 Kahden yhteenlasketun funktion plottaus

```
(define MAX_Y 2.4)

(define (draw world)
  (local ((define (f x) (sin (+ x world)))
          (define (g x) (sin (* 2(+ x world))))
          (define (h x) (+ (f x) (g x))))
    (plot h (- MAX_X) 0.2 (empty-scene WIDTH HEIGHT))))
```

Tehtävä

Piirrä myös x- ja y-akselit ja niiden asteikot.

Tehtävä vielä

Asteikko? Jakomerkit? Mikä on oikea termi. Ticks englanniksi.

3.30 Lukujonot

```
(define (jono aloitus operaatio askel määrä)
  (cond [(= määrä 0) empty]
        [else (cons aloitus (jono (operaatio aloitus askel) operaatio askel (- määrä 1)))]))

(check-expect (aritmeettinen-jono 0 2 5) (list 0 2 4 6 8))
(check-expect (aritmeettinen-jono -7 3 4) (list -7 -4 -1 2))
(define (aritmeettinen-jono aloitus askel määrä)
  (jono aloitus + askel määrä))

(check-expect (geometrinen-jono 0 10 5) (list 0 0 0 0 0))
(check-expect (geometrinen-jono 2 2 5) (list 2 4 8 16 32))
(check-expect (geometrinen-jono 2 1/2 5) (list 2 1 1/2 1/4 1/8))
(define (geometrinen-jono aloitus askel määrä)
  (jono aloitus * askel määrä))

(check-expect (fibonacci 1) 1)
(check-expect (fibonacci 2) 1)
(check-expect (fibonacci 3) 2)
(check-expect (fibonacci 4) 3)
(define (fibonacci mones)
  (cond [(or (= mones 1) (= mones 2)) 1]
        [else (+ (fibonacci (- mones 1)) (fibonacci (- mones 2)))]))
```

LUKU 4

Ohjelmoinnin ydinaiheita

LUKU 5

Ohjelmoinnin valinnaisia aiheita

- Tiedostojen luku ja kirjoittaminen
- Tiedostojen käsittely listoina
- Tietotyypit
- Liukuluvut
- Tietokoneen rakenne
- Tietoverkot

LUKU 6

Ohjelmoinnin sovelluksia

Tehtävä vielä

Piirtokomentojen kertauskappale, jossa selitetään ainakin overlay/xy:n ja overlay/offsetin ero.

Tehtävä vielä

Linkit kappaleen alkuun esitietovaatimuksiin ja niiden kertauksiin.

Tehtävä vielä

Kellotaulutehtävä

Tehtävä vielä

Tehtäville voisi luoda oman komennon, jotta niistä saa oman listauksen

Tehtävä vielä

Minne mallivastaukset?

Tehtävä vielä

Racketin builtinit saisi linkattua dokkareihin käyttämällä pygmentsin formatteria, joka wrappaisi lexeriltä tulevat keywordit anchoreiksi ja antaisi ne vasta sitten eteenpäin tavalliselle HtmlFormatterille.

Mikäli haluaisi vielä viilata tarkemmin, pitäisi tehdä uusi lexeri, jossa keywordit ja builtinit olisi korvattu BSL:n vastaavilla muokkaamalla tests/examplefiles/example.rkt ulostuloa

Tehtävä vielä

Linkit tehtävistä yms. aiheista OPS:n. Jokin systeemi, jolla taas OPS-osassa näkee mitkä tehtävät kattavat minkäkin kohdan, ja mitä jää kattamatta.

Tehtävä vielä

Graafi, josta näkee missä järjestyksessä kappaleet pitää käydä. Esim. Gephi voi saattaa tehdä nättejä, jos Graphviziä ei saa säädettyä.

Tehtävä vielä

Algoritmisen ajattelun käsittely

Tehtävä vielä

Ohjelmoijien metakognitiiviset prosessit? Miten lähteä luomaan ohjelmaa. Design recipe auttaa, mutta ei välttämättä tarpeeksi.

Tehtävä vielä

handinilla tehtävien tarkastusta

Tehtävä vielä

EV3-kirjasto, ja drracket plugini helpompaan käyttämiseen

Tehtävä vielä

Kuvat on Safarilla valtavia

LUKU 7

Ideoita

- “Vahvistetaan ymmärrystä tarkan arvon ja likiarvon erosta sekä pyöristämisestä.”
- “Opitaan neliöjuuren käsite ja käytetään neliöjuurta laskutoimituksissa” Pythagoras kaikissa etäisyyttä vaativissa 2D- ja 3D-animaatioissa
- Livecoding-moduuli extemporella (musiikki)
- Livecoding-moduuli fluxuksella
- Robottimoduuli EV3:lla
- binäärihaku, esimerkiksi funktion nollakohdalle
- jotain numeeristä integrointia trapezoideilla?
- Graafihaku vaikka shakkitehtävä siirroista?
- https://projecteuler.net/problems
- Vaikeaa, mutta parseri joka ymmärtää polynomeja

Symbols

A

B

C

D

E

F

G

H

I

K

L

M

N

P

R

S

T

V

Y