CS-A1120 Ohjelmointi 2

Aalto-yliopisto
Perustieteiden
korkeakoulu

Luento 2
kevät 2026

Sanna Suoranta (suomeksi, AS2) ja
Lukas Ahrenberg (englanniksi, T1)

2.3.2026

https://presemo.aalto.fi/o2fi2026

Bitit ja data

https://presemo.aalto.fi/o2fi2026

Aalto-yliopisto
Perustieteiden
korkeakoulu

Moduuli I – Tietokoneen salaisuudet

- **Tavoite: tietokoneen peruspalikat – alhaalta ylöspäin,**
  - Kuinka tietoa esitetään
  - Kuinka rakennetaan logiikkapiirejä, jotka käsittelevät dataa
  - Kuinka käyttää kellotettua palautetta piireissä
  - Kuinka ohjelmoida ilman korkean tason ohjelmointikieltä (assembly-kielellä)

- **Rakennamme tietokoneen käyttäen Scalaa**
  - Teemme loogisia komponentteja esittävät luokat
  - Lopulta työskentelemme simuloidulla suorittimella (prosessorilla, processor)

Viime viikon laskuesimerkki uudestaan

```
scala> def test(m: Int) = (m * (m + 1))/2
def test(m: Int): Int

scala> test(3)
val res: Int = 6

scala> test(50000000) // 5*10^7
val res: Int = -814376896


scala> def test(m: Long) = (m * (m + 1))/2
def test(m: Long): Long

scala> test(50000000) // 5*10^7
val res: Long = 1250000025000000

scala> test(50000000000)  // 5*10^10
val res: Long = -4378596987249509888
```

Viime viikon laskuesimerkki uudestaan

```
scala> def test(m: Int) = (m * (m + 1))/2
def test(m: Int): Int

scala> test(3)
val res: Int = 6

scala> test(50000000) // 5*10^7
val res: Int = -814376896
```

```
scala> def test(m: Long) = (m * (m + 1))/2
def test(m: Long): Long

scala> test(50000000) // 5*10^7
val res: Long = 1250000025000000

scala> test(50000000000)  // 5*10^10
val res: Long = -4378596987249509888
```

Huomioita

- **`Int`: virhe pienemmällä luvulla**
- **`Long` sallii suuremmat luvut**
- **Mutta molemmat ovat äärellisiä**
- **Kuten ehkä tiedät**
  - `Int` on 32 bittiä pitkä
  - `Long` on 64 bittiä pitkä
- **Mutta miksi negatiivisia lukuja?**

Tämän luennon jälkeen

- **Olet tutustunut binääri- ja heksadesimaalilukuihin (binary and hexadecimal numbers)**
- **Ajattelet tietoa bittisekvenssinä ja tunnet joitakin perusdatatyyppejä**
- **Tiedät Boolen-operaatioiden (Boolean operation) syntaksin ja siirto-operaattorit (shift operators) kokonaislukutyypille Scalassa**
- **Olet tietoinen IEEE 754 liukulukustandardista**

- **Kurssimateriaalissa vielä lisää yksityiskohtia!**

Analoginen ja digitaalinen

- **Analooginen käyttää jatkuvaa ilmiötä**
- **Digitaalinen käyttää äärellistä joukkoa tiloja**

Antikythera-laite
(kuva: wikipedia CC 2.5. Generic)

https://en.wikipedia.org/wiki/Antikythera_mechanism

- **Vaikka analogisia tietokoneita on ollut (sekä historiassa että nykyään), kaupalliset tietokoneet ovat nykyään digitaalisia**

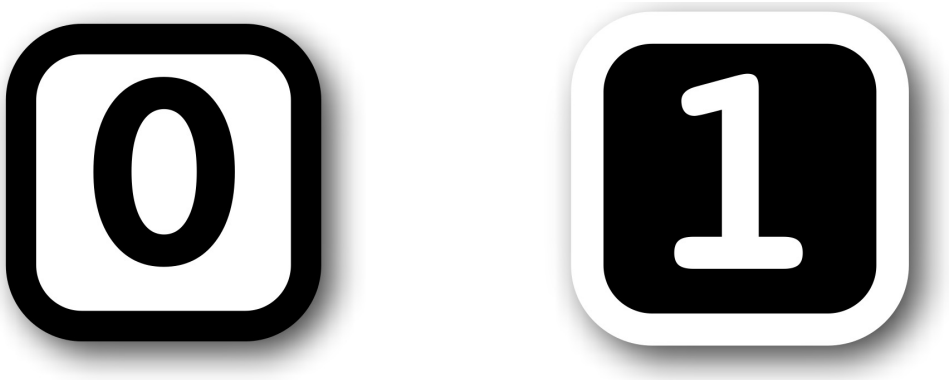
Googlen datakeskus
(kuva https://www.google.com/about/datacenters/gallery/)

Numerot (Digits)

- **Digit (englannin kielessä) = sormi / varvas**
- **10-kantaisessa lukujärjestelmässä on kymmenen numeroa:**
  - 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
  - (decimal number system)
- **Binäärilukujärjestelmässä on kaksi:**
  - 0, 1

Bitit (Bits)

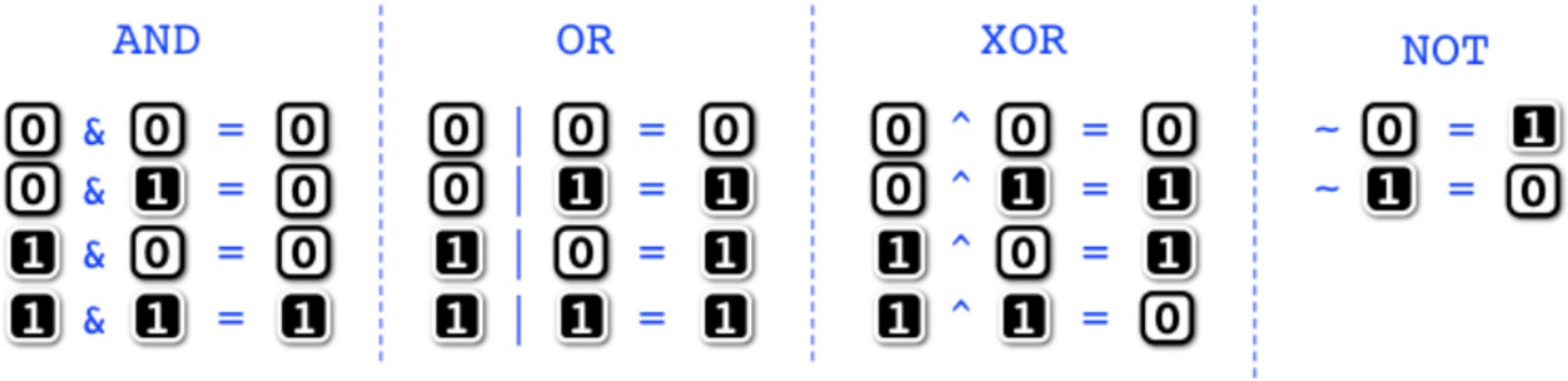
- **Bitti (bit), lyhennys sanoista binääriluku (binary number)**
  - Yleensä käytetään lukuja 0 ja 1, mutta symboleilla ei ole väliä
- **Informaation esittämisen perusyksikkö**



Claude E. Shannonin [1] mukaan J. W. Tukey keksi sanan "bit" esittämään lukua 0 tai1 binäärilukujärjestelmässä.

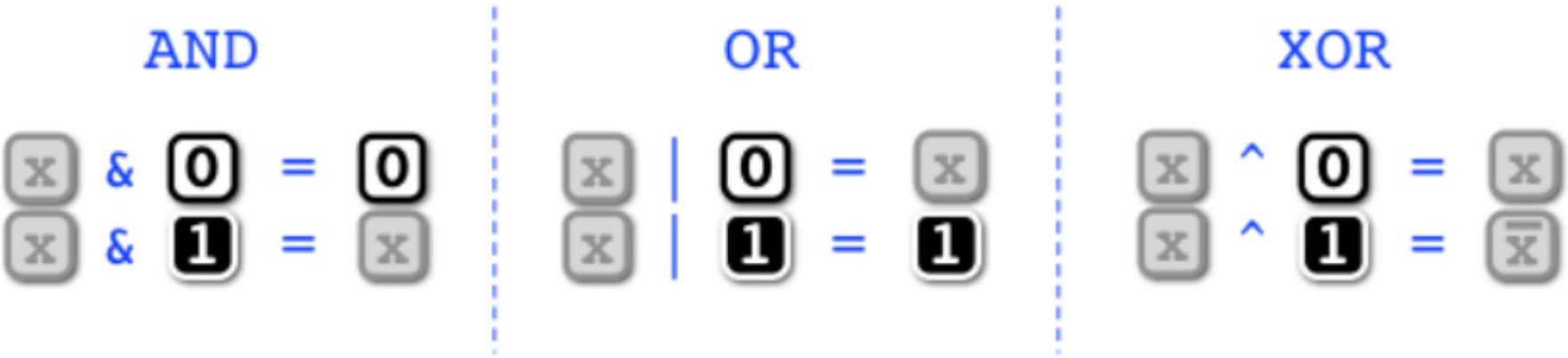
[1] C. E. Shannon. (1948). A mathematical theory of communication. The Bell System Technical Journal (vol 27, Issue 3, July 1948) DOI: 10.1002/j.1538-7305.1948.tb01338.x https://ieeexplore.ieee.org/document/6773024

Boolen-operaatiot biteille



- JA (AND): “minimi”
- TAI (OR): “maksimi”
- EI (NOT): käännä bitti (toggle)
- Poissulkeva-TAI (XOR, eXclusive OR): “summa” tai “pariteetti” (eli onko ykkösiä binääriluvussa parillinen vai pariton määrä)

Bitin arvon pakottaminen tai vaihtaminen



Data

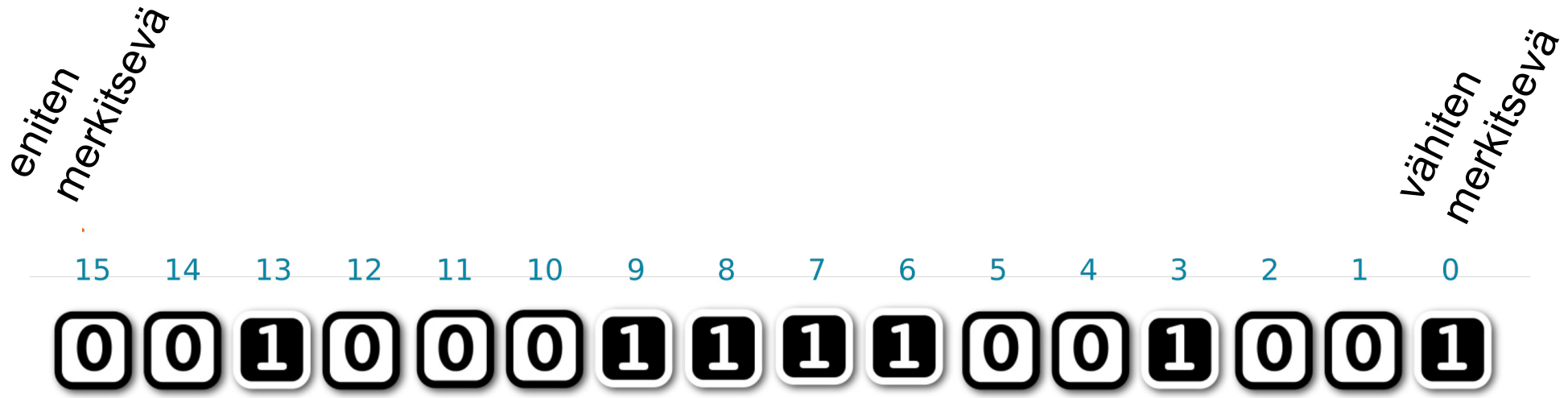
- **Laskennassa tietoa (data) esitetään bittijonona**
- **Perusmittayksikkö datan määrälle on numerojonon bittien lukumäärä**

Sana, sanan pituus

- **Tietokoneet on rakennettu käsittelemään dataa tietynmittaisten bittijonojen muodostamien yksiköiden avulla (esim. 8, 16, 32, 64 tai 128 bittiä)**
- **Tätä yksikkö kutsutaan sanaksi (word)**
- **Prosessoriarkkitehtuurissa sanan pituus (word length) on suurin määrä bittejä, jolla suoritin voi tehdä laskentaa yhdellä käskyllä**
- **Eli 64 bitin suoritin tyypillisesi tarkoittaa, että sanan pituus on 64 bittiä.**
  - Suurempi data pilkotaan sanan mittaisiksi sekvensseiksi sanoja tietokoneen muistissa

Bittien indeksit sanassa

- **Esimerkkinä 16-bittinen sana**



- **Samoin kuin desimaaliluvuissa, oikealla on vähiten merkitsevä luku**

Bitit erottavat asioita

- **Yksi bitti mahdollistaa kahden eri tilan (asian) erottamisen toisistaan (represent, tell apart)**
  - Jos 0 = 🍒 ja 1 = 🍍 , voimme kertoa eron kirsikoiden ja ananasten välillä
- ***n*:llä bitillä voimme esittää 2^n erilaista tilaa**
- **Binäärijono voidaan tulkita binääriluvuksi**

Bitit erottavat asioita

- **Eli kahdella bitillä voimme erottaa neljä tilaa**

bits	
00	= 🍒
01	= 🍍
10	= 🍏
11	= 🍉

bits	
00	= a
01	= b
10	= c
11	= d

bits	
00	= 0
01	= 1
10	= 2
11	= 3

- **`n`:llä bitillä saadaan 2^n merkkijonoa, joiden pituus on `n`**
  - `Int` on 32 bittiä, eli 2^{32} eri tilaa
  - `Long` on 64 bittiä, eli 2^{64} eri tilaa

Luonnollisten lukujen (integers) esittäminen

- **Luku “754” voidaan esittää usealla tavalla:**
  - Roomalaisin luvuin: DCCLIV
  - Kymmenlukujärjestelmässä: 754
  - Binäärilukuna: 1011110010
- **Kymmen- ja binäärilukujärjestelmä ovat esimerkkejä paikkajärjestelmästä (positional number system), mutta niillä on eri kantaluku (base, radix)**
  - $(2025)_{10} = 2*10^3 + 0*10^2 + 2*10^1 + 5*10^0$
  - Kantaluku voidaan merkitään luvun perään, sen alanurkkaan, jos sekaannus voi uhata, esim. $(11)_2$ tai $(11)_{10}$

Paikkajärjestelmät

- **Paikkalukujärjestelmässä meillä on joukko symboleita**
- **Joukon kokoa kutsutaan kantaluvuksi (base, radix)**
  - Desimaalilukujärjestelmässä luvut ovat 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, eli näitä on 10, joten kantaluku on 10
  - Binäärilukujärjestelmässä on luvut 0, 1 ja kantaluku on 2
  - Ohjelmoidessa käytetään myös heksadesimaalilukujärjestelmää, jossa käytetään yleensä symboleita 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F, jolloin kantaluku on 16
- **Huom. Symbolit itsessään ei ole tärkeitä**
  - `n`-kantaisessa järjestelmässä tarvitaan `n` erilaista symbolia (tilaa). Ei ole väliä, onko ne 0-9, A-J, binääriluvut 0 tai 1, tai vihreä ja sininen

Paikkajärjestelmän numeroiden tulkinta

- **Kirjoitamme luvut merkkijonoina, joissa vähiten merkitsevä numero (least significatn digit) on oikeanpuoleisimpana ja eniten merkitsevä (most significant) vasemmanpuoleisimpana**
- **Paikan arvo lasketaan b^i missä b on kantaluku ja i indeksi alkaen nollasta vähiten merkitsevälle luvulle**
  - Esim. $754_{10} = 700 + 50 + 4 = 7*10^2 + 5*10^1 + 4*10^0$
  - Sama tulkinta sopii binäärilukujen muuttamiseen kymmenkantaiseksi:

$$
\begin{aligned}
(1011110010)_2 &= 1*2^9 + 0*2^8 + 1*2^7 + 1*2^6 + 1*2^5 \\
&\quad + 1*2^4 + 0*2^3 + 0*2^2 + 1*2^1 + 0*2^0 \\
&= 512 + 128 + 64 + 32 + 16 + 2 \\
&= 754
\end{aligned}
$$

Binääriluvusta desimaaliluvuksi

- **Sama ”kaava” sopii binääriluvuille**
- **Algoritmina:**
  - Input: kaksikantainen luku $x \in \mathbb{Z}_{\geq 0}$
  - Alusta tulos arvoksi 0
  - Toista jokaiselle bittipositiolle $j \geq 0$
    - Jos bitti j on 1, niin
    - lisää tulokseen 2^j
- **Esimerkiksi** $11001_2 = 2^0 + 2^3 + 2^4 = 1 + 8 + 16 = 25$

Desimaaliluvusta binääriluvuksi (algoritmi)

- **Input: 10-kantainen luku `x` $\in \mathbb{Z}_{\geq 0}$**
- **Alusta tuloksen `res` kaikki bitit arvoksi `0` (`off`)**
- **Toista kunnes `x = 0`**
  - Etsi suurin mahdollinen $y = 2^j$ jossa $x - 2^j \geq 0$
  - Aseta bitti j olemaan `1` (`on`)
  - Päivitä x olemaan $x - y$
- **Esim. 25 binääriksi**

1. **Aluksi x = 25, res = 0**
2. **Etsi suurin $2^j \leq 25$**
   - Se on $y = 2^4 = 16$, eli $j = 4$
   - Aseta bitti 4 ykköseksi: res = 10000_2
   - Päivitä: $x = 25 - 16 = 9$
3. **Etsi suurin $2^j \leq 9$**
   - Se on $y = 2^3 = 8$ eli $j = 3$
   - Aseta bitti 3 ykköseksi: res = 11000_2
   - Päivitä $x = 9 - 8 = 1$
4. **Etsi suurin $2^j \leq 1$**
   - Se on $y = 2^0 = 1$ eli $j = 0$
   - Aseta bitti 0 ykköseksi: res = 11001_2
   - Päivitä $x = 1 - 1 = 0$
5. **X = 0 eli loppu. Tulos on $(11001)_2$**

Binääriluvuilla

- **Laskenta on helppoa binäärilukujärjestelmässä**
  - Vain kaksi tilaa, jotka koneen pitää erottaa toisistaan
  - Kertolasku on todella helppoa (ei tarvi muistaa taulukoita)

- **Ohjelmoijan tulee tuntea sekä binääriluvut että heksadesimaaliluvut**
  - Tietokoneiden arkkitehtuuri perustuu näille lukujärjestelmille, joten niillä laskenta on tehokasta

Hyvä tietää: etumerkilliset numerot (eli positiiviset ja negatiiviset luvut)

- **Etumerkki (sign) pitää esittää jotenkin**
- **Useita koodaustapoja, esimerkiksi (muitakin on):**
  - Etumerkki + suuruus (merkitsevin bitti on etumerkki)

    $2_{10} = 0010_2$; $-2_{10} = 1010_2$
  - Yhden komplementti (one's complement) – negatiiviset luvut saadaan positiivisista bittikohtaisella not-operaattorilla

    $2_{10} = 0010_2$; $-2_{10} = 1101_2$
  - Kahden komplementti (two's complement) – negatiiviset luvut saadaan positiivisista bittikohtaisella not-operaattorilla + 1

    $2_{10} = 0010_2$; $-2_{10} = 1110_2$
  - Kahden komplementtia käytetään yleisesti ohjelmointikielissä, sillä se tekee positiivisten ja negatiivisten lukujen yhteenlaskusta helpon

Kahden komplementin komen bitin numeroille

Bits	Signed two's comp.	Unsigned
000	0	0
001	1	1
010	2	2
011	3	3
100	-4	4
101	-3	5
110	-2	6
111	-1	7

- **Puolet mahdollisista tiloista käytetään negatiivisten lukujen esittämiseen**
- **Esitettävien arvojen suurin suuruus puolittuu**

Scalan perustietotyyppien pituudet bitteinä

- **Scala on alustariippumaton (hardware-independent) kieli, eli se on suunniteltu käytettäväksi monentyyppisillä suorittimilla (processor)**
- **Perustietotyyppien pituudet bitteinä Scalassa ovat**
  - Byte: 8 bittiä
  - Short: 16 bittiä
  - Int: 32 bittiä
  - Long: 64 bittiä
- **Koska bittejä on rajallinen määrä, yhteen- ja kertolasku voi johtaa ylivuotoon (overflow), jos lopputulos on liian suuri. Tämä kannattaa tiedostaa, koska se ei yleensä aiheuta virhetilannetta, mutta lopputulos voi olla odottamaton**

Numeerinen ja tekstuaalinen data

- **Useimmissa ohjelmissa on vähintään numeraalista (numeroita esittävää) ja tekstuaalista (tekstiä esittävää) dataa**
- **Perustietotyypit Scalassa voidaan jakaa näihin kahteen kategoriaan**
  - Numeraaliset, esittävät etumerkillisiä lukuja (eli + / -)

    `Byte, Short, Int, Long, Float, Double`

    - `Float` (32 bittiä), `Double` (64 bittiä)
  - Tekstuaaliset, esittävät merkkejä ja merkkijonoja

    `Char, String`
  - (`String` on oikeastaan `Char`-sekvenssi, `Char` on 16 bittiä pitkä)

Mitä data esittää?

- **Bittijono on vain nollien ja ykkösten merkkijono**
- **Kiinnostavaa on se, mitä tuo bittijono esittää standardien ja sopimusten mukaan**

29058 bittiä

- **Jos meillä olisi seuraava 29058 bitin jono (kaikki bitit ei mahdu tähän, mutta jono alkaa seuraavasti)**

11111111111111111111110001111101111111111111111111111111111111111111111111111

- **Tämä voitaisiin tulkita todella suureksi kokonaisluvuksi, tai…**

29058 = 167 * 174 bittiä

- **Jaettuna bittijono 175 riviin, jossa jokaisessa on 167 saraketta, saadaan tällainen**

jatkuu

29058 = 167 * 174 bittiä

- **Jos tulkitsemme, että 0 on musta ja 1 on valkoinen, tulos on**

https://xkcd.com/371/

Tiedon muoto

- **Tulkinnassa on kyse standardeista, muodosta ja käytänteistä**
  - Bittijono (data) voidaan tulkita monella tapaa (mutta kaikki tavat eivät tuota mitään järkevää)
- **Tarvitsemme käytänteitä (conventions) datan esittämiselle. Näitä kutsutaan formaateiksi (formats) tai**
- **(ohjelmointikielisessä) niihin viitataan tietotyyppeinä (data type)**

0100000000001001001000011111101101010100010001000010110100011000

  - Tulkittuna liukulukuna (Double) Scalassa on
    3.14159265358979311599796346854418516159057617187 5
  - Tulkittuna pitkänä kokonaislukuna (Long) Scalassa se onkin
    4614256656552045848

Seuraavat 64 bittiä liukulukuna

0100000000001001001000011111101101010100010001000010110100011000

- **Liukulukustandardin IEEE Std 754 mukaan tulkittuna yllä oleva bittijono on**

  3.14159265358979311599796346854418516159057617187 5

- **Tämän luvun saat, kun käytät Scalan sisäänrakennettua vakiota**

  `scala.math.Pi`

- **Ei ihan matemaattisen eksakti vakio π mutta aika hyvä**

Liukuluvut (floating point numbers)

- **Mitä tässä oikein tapahtuu?**

```
scala> 0.3 - 0.1
val res10: Double = 0.19999999999999998
```

Liukulukustandardi

- **IEEE Std 754 (https://ieeexplore.ieee.org/document/8766229)**
  - `Float` = IEEE 754 ‘binary32’
  - `Double` = IEEE 754 ‘binary64’
- **Äärellinen määrä bittejä → kaikkia rationaalilukuja ei voi esittää**
  - Se mitkä luvut voidaan esittää äärellisellä määrällä lukuja riippuu kantaluvusta
- **IEEE 754-2008 ‘binary64’ liukulukujen lähimmät arvot:**
  - $0.3 \approx (1.0011001100110011001100110011001100110011001100110011)_2 \times 2^{-2}$ $= 0.2999999999999998889776975374843459576368319091796875$
  - $0.1 \approx (1.1001100110011001100110011001100110011001100110011010)_2 \times 2^{-4}$ $= 0.1000000000000000055511151231257827021181583404541015625$

Mitä on 0,3 – 0,1 standardin IEEE 754 mukaan?

- **Oikeasti tietokone laskee lukujen binääriesitysten avulla 0,3 – 0,1= 0.29999999999999988897769753748434595763683319091796875 −0.10000000000000005551115123125782702118158340454101562 5**
- **Kannattaa olla tietoinen liukulukujen ominaisuuksista kun niitä käyttää**
  - Tutustu standardiin
  - Muista, kun teet laskentaa kiinteällä joukolla rationaalilukuja, että ne harvenevat sitä mukaa kun niiden arvo on kauempana nollasta (Lisää kurssimateriaalissa: https://a1120.cs.aalto.fi/2024/notes/round-bits-and-data--numerical.html)

Seuraavat 88 bittiä tekstinä

01001111011010110110000101111001001011000010000001101000011101010110110101100001011011110

- **Jaettuna 8 bitin ryhmiin ja tulkittuna Unicode-standardin (UTF-8) mukaisesti, saamme tekstin**

```
Okay, human
```

Seuraavat 32 bittiä

10000010111010011110000011000100

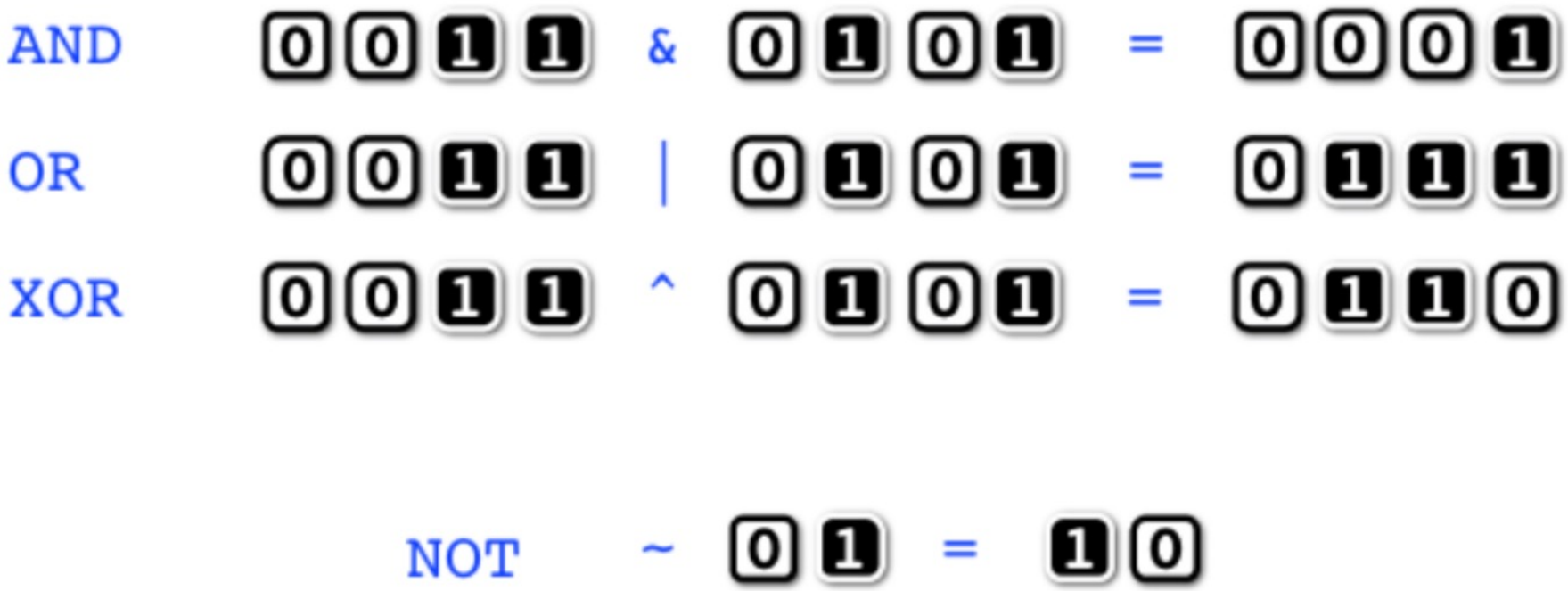
- **Tulkittuna Internet-standardin RFC 791:n(ja RFC 796:n) mukaan, tämä on B-luokan Internet-protokollan osoite (IPv4-osoite). Jaettuna 8 bitin ryhmiin, joista kukin erikseen muutettuna kymmenjärjestelmään, saadaan ihmisille (ehkä) tutumpi muoto:**

  130.233.224.196

- **ja katsottuna nimipalvelusta (Domain Name System, DNS) kyseistä osoitteesta löytyvän koneen nimi on**

  kosh.aalto.fi

Boolen-operaatiot sanoille



- **Yleisen tavan (engl. convention) mukaan bittioperaatiot toimivat riippumattomasti jokaiselle sanan bittipositiolle**

Esimerkki IP-osoitteen verkko-osasta

10000010111010011110000011000100 eli 130.233.224.196

- **Internet Protokollan (IP) osoite toimii osoitteena, ja alkuosa kertoo verkon osoitteen ja loppuosa toimii koneen tunnisteena**
- **Verkon osoitteen saa verkkopeitteen (network mask) avulla, kun tietää kuinka monta bittiä kuuluu verkko-osaan:**
  - 130.233.224.196/25

	10000010111010011110000011000100	osoite
AND	11111111111111111111111110000000	/25
	10000010111010011110000010000000	verkko

  - Tähän verkkoon mahtuu $2^{(32-25)} = 2^7$ => 126 konetta (+ verkon itsensä osoite eli pelkkiä nollia ja yleislähetysosoite ykkösiä)

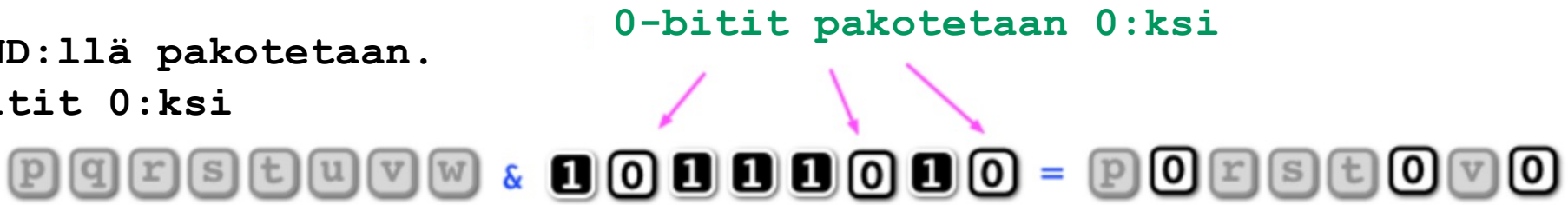
Jos kaikki data tietokoneessa esitetään käyttäen binäärilukuja, sitten kaikkien tietotyyppejä käyttävien operaatioiden pitää olla rakennettu bittimanipulaatioiden avulla

Sanan arvon pakottaminen tai vaihtaminen

Aloitetaan sanan biteillä:

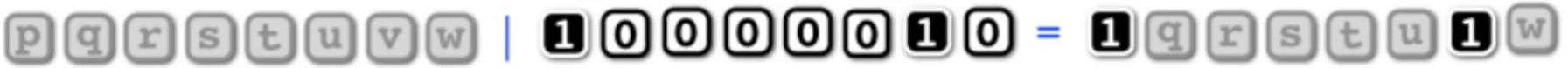
AND:llä pakotetaan.
bitit 0:ksi

0-bitit pakotetaan 0:ksi



OR:lla pakotetaan.
bitit 1:ksi

1-bitit pakotetaan 1:ksi



XOR:n avulla vaihdetaan
bitin arvo

1-bitti vaihdetaan

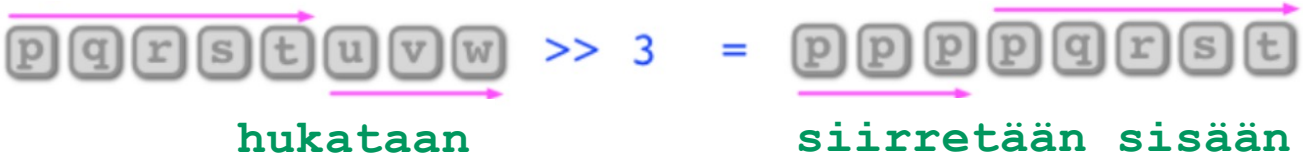
R on vaihdettu

Sanan bittien siirto (shifting)

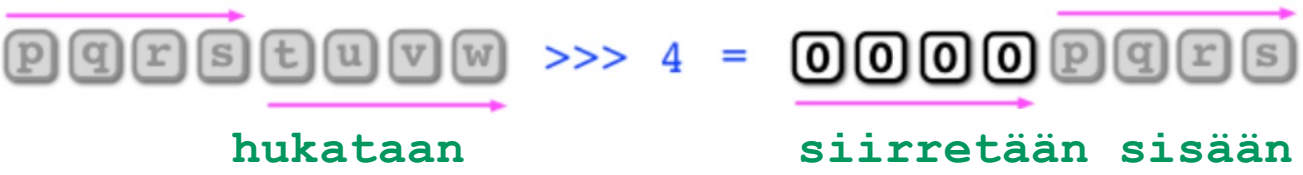
Aloitetaan sanan biteillä:

Vasen siirto (left shift)

Oikea aritmeettinen siirto (right arithmetic shift)



Oikea looginen siirto (right logical shift)



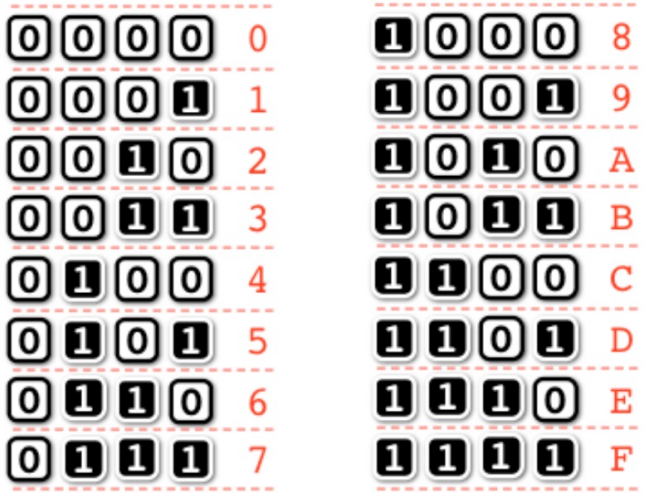
Heksadesimaaliluvut

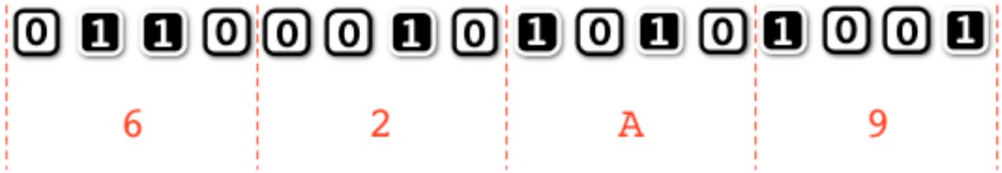
- **Kantaluku 16, luvut 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F**
- **Ohjelmoijan olisi syytä tuntea nämä**
  - Helppoja muuttaa binääriesitykseksi
  - Kompakti (ja yksilöivä) tapa esittää
- **Neljä bittiä on yksi heksadesimaaliluku**
  - $2^4 = 16$
- **(tietokoneen) sanat ovat yleensä neljän kerrannaisia (16, 32, 64)**

Binary	Hexadecimal
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

Heksadesimaaliluvut – esimerkki

- **Yhden heksadesimaaliluvun esittämiseen tarvitaan 4 bittiä**
- **Esimerkiksi kymmenjärjestelmän luku 25257 on heksadesimaalilukuna 62A9**





Esimerkki: IPv6-osoite heksadesimaalilukuna

`2a03:4e42:0:0:0:0:0:347`

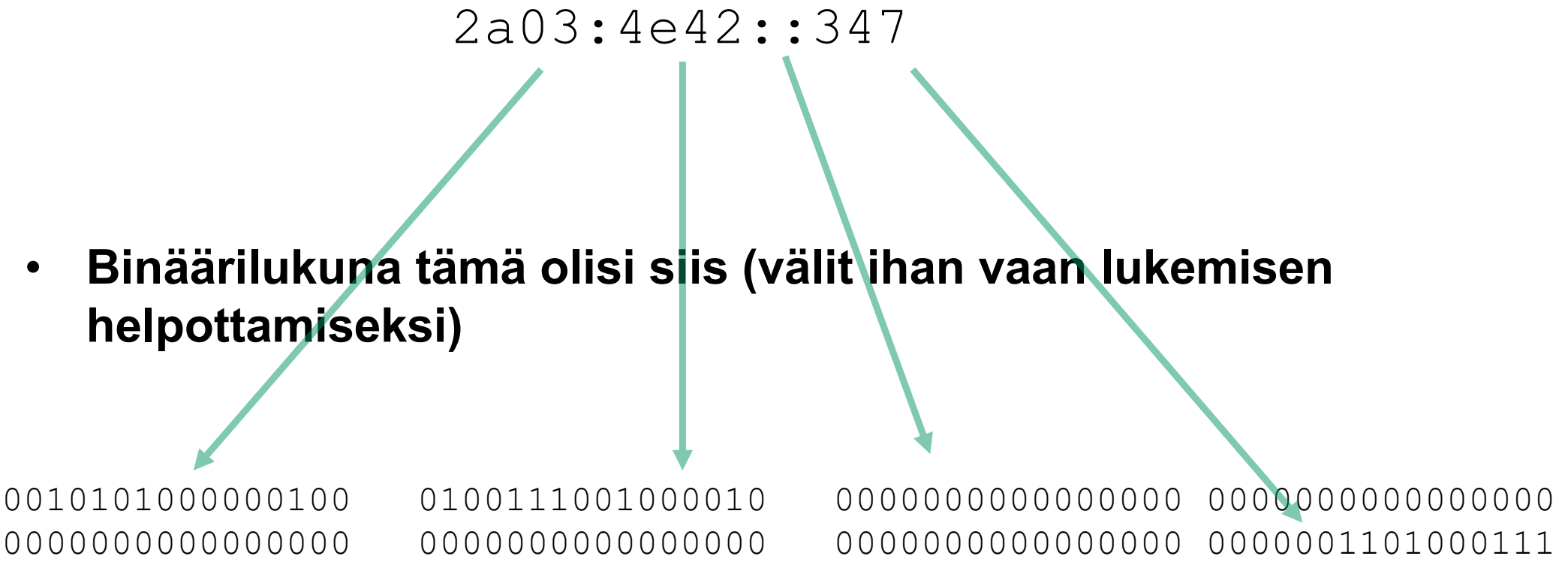
- **Yllä on Internet-starndardin RFC 2373 mukainen IPv6-osoite palvelimelle www.aalto.fi (http-protokollan pyyntöön se vastaa web-sivulla), 8:n ryhmissä eli `0 - ffff`**
  - IPv6-osoitteet ovat 128 bittiä pitkiä, joten ne on kätevintä esittää heksadesimaaliluvuilla kymmenjärjestelmän lukujen sijaan
  - Yleensä peräkkäiset 0-arvot korvataan :: eli tämä olisi oikeastaan

    `2a03:4e42::347`

  - (Jos esitettäisiin samoin kuin IPv4-osoite kymmenjärjestelmän lukuna, tämä olisi

    42.4.78.66.0.0.0.0.0.0.0.0.0.0.3.71

Esimerkki: IPv6-osoite heksadesimaalilukuna



- **Binäärilukuna tämä olisi siis (välit ihan vaan lukemisen helpottamiseksi)**

Heksadesimaaliluvut Scalassa

- **Useimmat ohjelmoinkielet tukevat heksadesimaaliliteraaleja laittamalla niiden eteen prefiksin `0x`**
  - Esim `0x62A9` on heksadesmiaaliluku `62A9`

```
scala> val a = 0x62A9
val a: Int = 25257
```

- **Huomaa että `a` on tyyppiä `Int`, ja se tulostetaan kuin desimaaliluku Scala REPLissä**
- **Saamme sen heksadesimaaliesityksenä (sisältäen `0x`) formatoidulla merkkijonolla**

```
scala> print("0x%X".format(a))
0x62A9
```

  - Tässä `%X` tarkoittaa “ota format-käskyn syöte ja esitä se heksana”
- **Muissa kielissä muitakin (esim. 8-kantainen octal `0o` tai binäärille `0b`)**

Esimerkki: tietyn paikan bitin tarkistus Scalassa

- **Siirrä bittiä oikealle kunnes se on vähiten merkitsevän asemassa**
- **Käytä bittikohtaista (bit-wise) AND ykkösen kanssa**
  - jos tulos on 1, niin bitti on asetettu (ykköseksi)

```
scala> val x = 0x3A // 111010 in binary
val x: Int = 58

scala> (x >> 0) & 1 // value of bit at position 0
val res0: Int = 0

scala> (x >> 1) & 1 // value of bit at position 1
val res1: Int = 1
```

- **(hassuna) funktiona:**

```
def isOn(x:Int, j:Int) =
  if ((x>>>j) & 1) == 1
  then println(s"Bit $j is ON in $x")
  else println(s"Bit $j is OFF in $x")
```

Esimerkki: luo sana, jossa on 1 tietyssä paikkaa ja 0 muualla

- **Joskus on näppärää luoda sana, jossa on nollat kaikkialla muualla paitsi jossain tietyssä paikassa `j`**
- **Tämä voidaan tehdä siirtämällä vasemmalle `1` tarvittava määrä askelia**
  - Muista tyyppi! `1` on 32 bittiä ja `1L` on 64 bittiä

Esimerkki: luo sana, jossa on 1 tietyssä paikkaa ja 0 muualla – 32-bittinen

```
scala> val j = 3 // Index 3

val j: Int = 3

scala> val a = 1<<j // Take a 1 and shift it left the required number of steps

val a: Int = 8

scala> tellAboutInt(a)

I am a 32-bit word that Scala views as having format 'Int' ...

... my bits are, in binary, (00000000000000000000000000001000)_2

... or equivalently, in hexadecimal, 0x00000008

... Scala prints me out as 8

... I represent the signed decimal integer 8

... I represent the unsigned decimal integer 8
```

Esimerkki: luo sana, jossa on 1 tietyssä paikkaa ja 0 muualla – 64-bittinen

```
scala> val j = 55 // Index 55

val j: Int = 55

scala> val c = 1L<<j // If you need 64 bits use a Long integer

val c: Long = 36028797018963968

scala> tellAboutLong(c)

I am a 64-bit word that Scala views as having format 'Long' ...

... my bits are, in binary,
(0000000010000000000000000000000000000000000000000000000000000000)_2

... or equivalently, in hexadecimal, 0x0080000000000000

... Scala prints me out as 36028797018963968

... I represent the signed decimal integer 36028797018963968

... I represent the unsigned decimal integer 36028797018963968
```

Materiaalissa: Bittien paljastaminen Scalassa

- **Voit käyttää `tellAbout...` -funktiota, jotka on annettu kurssin materiaalissa**
- **Tulostavat binääriesityksen Scalan perusdatatyypeistä ja merkkijonoista**
- **Copy-paste määritelmät Scala REPLiin tai koodiin**
- **Voivat auttaa harjoitustehtävien bugien etsinnässä**

- **`tellAboutByte`**
- **`tellAboutShort`**
- **`tellAboutInt`**
- **`tellAboutLong`**
- **`tellAboutChar`**
- **`tellAboutString`**
- **`tellAboutFloat`**
- **`tellAboutDouble`**

Oppimateriaalissa:
https://a1120.cs.aalto.fi/2024/notes/round-bits-and-data--witness.html

Mutta mitä jos tarvitsen todella isoja numeroita?

- **Scalassa on tietorakenne nimeltä `BitInt` joka voi käsitellä määrättömän kokoisia kokonaislukuja**

```
scala> def test(m: BigInt) = (m * (m + 1))/2
   def test(m: BigInt): BigInt

   scala> test(50000000000L)
   val res0: BigInt = 1250000000025000000000
```

- **Vastaavia on muissakin kielissä**
- **Huom. Aritmeettisten operaatioiden vaatima työmäärä saattaa riippua tallennetusta luvusta**
  - Ei enää vakioaikaista, koska ei mahdu yhteen “sanaan”
  - Lisää tästä tulee kierroksella 7

Suuruusluokka

- **Tiedon määrää voidaan mitata myös tavuina (byte)**
  - 1 tavu = 8 bittiä
- **Prefikseinä käytetään tavallisia kansainvälisen yksikköjärjestelmän (SI-yksiköt) prefiksejä (kilo, mega, giga jne) kertomaan suuruusluokasta (orders of magnitude)**
- **Suuruus on tyypillisesti annettu joko 10-kantaisena tai 2-kantaisena**

Mittayksiköt datalle

- **Tavuina**

Prefix	Short	Order
kilo	k	10^3
mega	M	10^6
giga	G	10^9
tera	T	10^{12}
peta	P	10^{15}
exa	E	10^{18}
zetta	Z	10^{21}
yotta	Y	20^{24}

- **Bitteinä**

Prefix	Short	Order
kibi	Ki	2^{10}=1024
mebi	Mi	2^{20}=1048576
gibi	Gi	2^{30}=1073741824
tebi	Ti	2^{40}=1099511627776
pebi	Pi	2^{50}=1125899906842624
exbi	Ei	2^{60}==1152921504606846976
zebi	Zi	2^{70}=1180591620717411303424
yobi	Yi	2^{80}=1208925819614629174706176

- **Huomaa, että isommissa suuruusluokissa on merkityksellistä, kumpaa näistä käytetään**

Tämän viikon harjoituksista

- **Käytä Boolen-operaattoreita Scalassa**
- **`Int`- tai `Long`- jne -tyyppisiä muuttujia ajatellaan bittien jonoina**
- **Muista että vähiten merkitsevä bitti on oikeanpuoleisin**
- **Opettelet, kuinka siirtää (shift), tarkistaa (check) ja asettaa (set) bittejä**
  - Oppimateriaalissa esimerkkejä!

Fukseille – Johdatus opiskeluun to 12.3!

- **Muistakaa Johdatus opiskeluun –kurssin luento to 12.3. klo 14:15**
  - Info: TU7, Maarintie 8, Valinnat uran aikana
  - Tepsy: maisteriohjelman valinta myöhemmin
  - Tfy: ?? U4, Otakaari 1, Valinnat uran aikana
  - Tik: T1, Konemiehentie 2, Akateeminen urapolku
  - Tuta, ei luentoa vaan jotain muita vaihtoehtoja

- **Sivuaineluento to 19.3 klo 14:15, Jeti, Otakaari 5**