# Ohjelmoinnin perusteet Y Python

T-106.1208

10.2.2010

# Kertausta: listat

- Tyhjä uusi lista luodaan kirjoittamalla esimerkiksi

  ```
  lampotilat = []
  ```

  (jolloin `lampotilat` on luotuun listaan viittaavan muuttujan nimi).
- Listan loppuun voi lisätä uuden arvon kirjoittamalla

  ```
  lampotilat.append(arvo)
  ```
- Jos listassa on jo vähintään `i+1` alkiota, voi indeksille `i` sijoittaa uuden arvon kirjoittamalla

  ```
  lampotilat[i] = arvo
  ```

  Tällöin indeksillä `i` ollut vanha arvo häviää.
- Indeksit ovat kokonaislukuja ja listan ensimmäisen alkion indeksi on aina 0.
- Listan yksittäistä alkiota voi käyttää esimerkiksi tulostuskäskyissä tai lausekkeissa:

  ```
  print "5. lampotila", lampotilat[4]
  summa = summa + lampotilat[4]
  ```

# Kertausta: listan läpikäynti

- While-käskyn avulla:

```
summa = 0
i = 0
while i < LKM:
    summa += lampotilat[i]
    i += 1
```

- For-käskyn avulla

```
summa = 0.0
for astemaara in lampotilat:
    summa += astemaara
```

Muuttuja `astemaara` saa arvokseen vuorotellen jokaisen listan alkion.

# Halutuimman arvon etsiminen listasta

- Usein toistuva ongelma on myös löytää listasta halutuin arvo, esimerkiksi suurin lämpötila.
- Käytetään apumuuttujaa, johon tallennetaan tähän asti löydetty paras arvo.
- Aluksi apumuuttujan arvoksi alustetaan joko mahdollisimman huono arvo tai listan ensimmäisen alkion arvo.
- Käydään listan alkiot läpi toistokäskyssä. Jos tällä kierroksella tarkasteltava alkio on parempi kuin apumuuttujan arvo, vaihdetaan sen arvo apumuuttujan arvoksi.
- On myös jotenkin käsiteltävä se tilanne, että lista on tyhjä.
- Seuraavan kalvon esimerkkifunktio etsii parametrina annetusta listasta suurimman lämpötilan.

# Maksimilämpötilan etsiminen, koodi

```
def etsi_maksimi(lampotilalista):
    if len(lampotilalista) == 0:
        return None
    else:
        maksimi = lampotilalista[0]
        for lampotila in lampotilalista:
            if lampotila > maksimi:
                maksimi = lampotila
        return maksimi
```

# Funktio `range`

- For-käskyn yhteydessä on käytetty funktiota `range` generoimaan haluttu lukujono.
- Täsmällisesti ottaen `range`-funktio palauttaa listan, joka sisältää kokonaislukuja.
- Esimerkiksi `range(7)` palauttaa listan `[0, 1, 2, 3, 4, 5, 6]` ja `range(3, 8)` listan `[3, 4, 5, 6, 7]`
- Funktiolle voi antaa myös kahden peräkkäisen alkion välin ja alkiot voivat myös pienentyä, esimerkiksi `range(15, 3, -3)` palauttaa listan `[15, 12, 9, 6]`.
- For-käskyssä

```
for i in range(6):
    kasky
```

tarkoittaa: suorita `kasky` jokaiselle `i`:n arvolle, joka kuuluu listaan `[0, 1, 2, 3, 4, 5]`

# Haku listasta: peräkkäishaku

- Usein toistuva ongelma: on tutkittava, löytyykö jokin arvo listasta ja jos löytyy, niin miltä indeksiltä.
- Jos luvut on tallennettu listaan mielivaltaisessa järjestyksessä, käytetään peräkkäishakua: käydään lista järjestyksessä läpi ja verrataan jokaista alkiota etsittävään arvoon.
- Läpikäyntiä jatketaan, kunnes haluttu arvo on löytynyt tai lista on käyty loppuun.
- Seuraavassa esimerkissä etsitään pistelistasta haluttu arvo. Mukana on myös funktio, joka ensin lukee pisteet listaan. Jos haluttu arvo esiintyy listassa monta kertaa, palautetaan niistä vain ensimmäisen indeksi.

# Peräkkäishaku: koodi

```
def lue_pisteet():
    rivi = raw_input("Monenko pisteet haluat antaa? ")
    lkm = int(rivi)
    pistelista = [0] * lkm
    i = 0
    while i < len(pistelista):
        rivi = raw_input("Anna seuraavan pelaajan pisteet: ")
        pistelista[i] = int(rivi)
        i += 1
    return pistelista

def hae_pisteet(lista, arvo):
    for i in range(0, len(lista)):
        if lista[i] == arvo:
            return i
    return -1
```

# Peräkkäishaku, koodi jatkuu

```
def main():
    pisteet = lue_pisteet()
    rivi = raw_input("Mita arvoa etsitaan? ")
    haettava_arvo = int(rivi)
    paikka = hae_pisteet(pisteet, haettava_arvo)
    if paikka == -1:
        print "Haettavaa arvoa ei loytynyt pistelistasta"
    else:
        print "Haettava arvo on listassa indeksilla", paikka


main()
```

# Kaikkien arvojen haku

- Jos listasta halutaan hakea kaikki halutun arvon esiintymät, voidaan niiden indeksit kerätä listaan ja palauttaa ko. lista.
- Toinen vaihtoehto on kirjoittaa haun tekevä funktio niin, että se ei aloita listan alusta, vaan parametrina annetusta indeksistä. Funktiota kutsutaan monta kertaa niin, että seuraavalla kutsukerralla aloitusindeksi on aina edellisellä kerralla löydettyä indeksiä yhden suurempi.

# Kaikkien esiintymien haku, koodi

```
def hae_pisteet_kaikki(lista, arvo):
    tuloslista = []
    for i in range(0, len(lista)):
        if lista[i] == arvo:
            tuloslista.append(i)
    return tuloslista
```

# Operaattori `in`

- Pythonissa on valmis operaattori `in`, jonka avulla voidaan tutkia, onko haettava arvo annetussa listassa:

```
if haettava_arvo in pisteet:
    print "Haettava arvo on pistelistassa"
else:
    print "Haettava arvo ei ole pistelistassa"
```

- Tai vaihtoehtoisesti:

```
if haettava_arvo not in pisteet:
    print "Haettava arvo ei ole pistelistassa"
else:
    print "Haettava arvo on pistelistassa"
```

- Operaattorin `in` käyttäminen ei ole kuitenkaan olennaisesti sen tehokkaampaa kuin listan läpikäyminen omalla funktiolla.

# Metodi `index`

- Pythonissa on myös valmis *metodi* `index`, jonka avulla voidaan hakea halutun alkion indeksi listasta.
- Metodi on kuin funktio, mutta sitä kutsutaan vähän eri tavalla:

```
listamuuttuja.metodi(parametrit)
```

- Esimerkki metodin `index` kutsumisesta:

```
paikka = pisteet.index(haettava_arvo)
```

- Tämä johtaa kuitenkin ohjelman kaatumiseen, jos haettavaa arvoa ei ole pistelistassa. Sen vuoksi kannattaa ensin tarkistaa, että arvo on listassa:

```
if haettava_arvo not in pisteet:
    print "Haettava arvo ei ole pistelistassa"
else:
    paikka = pisteet.index(haettava_arvo)
    print "Haettava arvo on listassa indeksilla", paikka
```

# Binäärihaku

- Jos tiedetään, että listassa olevat alkiot ovat suuruusjärjestyksessä, binäärihaku on selvästi tehokkaampi kuin peräkkäishaku.
- Periaate:
  1. Valitaan koko lista hakualueeksi.
  2. Lasketaan hakualueen keskimmäisen alkion indeksi.
  3. Verrataan keskimmäistä alkiota haettavaan alkioon. Jos ne ovat yhtäsuuret, haettava alkio on löytynyt, joten lopetetaan.
  4. Jos haettava alkio on pienempi, jatketaan kohdasta 2 niin, että hakualueena on alkuperäisen hakualueen alkupuolisko.
  5. Jos haettava alkio on suurempi, jatketaan kohdasta 2 niin, että hakualueena on alkuperäisen hakualueen loppupuolisko.

# Binäärihaku, koodi

```
def binaarihaku(lista, arvo):
    ala = 0
    yla = len(lista) - 1
    while ala <= yla:
        keski = (ala + yla) / 2
        if lista[keski] == arvo:
            return keski
        elif lista[keski] < arvo:
            ala = keski + 1
        else:
            yla = keski -1
    return -1
```

# Alilistat

- Listasta voi ottaa helposti alilistoja:

```
>>> lista = [2, 4, 6, 8, 10, 12, 14, 16]
>>> alilista = lista[2:5]
>>> print alilista
[6, 8, 10]
```

  Kaksoispisteen jälkeen annettua indeksia ei enää oteta mukaan.
- Ensimmäinen tai viimeinen indeksi voidaan myös jättää merkitsemättä, jolloin alilistaan otetaan alkiot listan alusta lähtien tai listaan loppuun asti:

```
>>> lista[:5]
[2, 4, 6, 8, 10]
>>> lista[5:]
[12, 14, 16]
```

- Negatiiviset indeksit tarkoittavat alkioita listan lopusta lähtien:

```
>>> lista[:-1]
[2, 4, 6, 8, 10, 12, 14]
```

# Lisäys listaan

- Listan loppuun voi lisätä uuden alkion append-metodilla:

```
>>> lista = [2, 4, 6, 8, 10, 12, 14, 16]
>>> lista.append(-3)
>>> print lista
[2, 4, 6, 8, 10, 12, 14, 16, -3]
```

- Sijoituskäsky ei lisää listaan uutta alkota, vaan se korvaa vanhan alkion uudella:

```
>>> lista[3] = -5
>>> print lista
[2, 4, 6, -5, 10, 12, 14, 16, -3]
```

- Sen sijaan metodilla `insert` voi lisätä halutulle indeksille uuden alkion. Lisäyspaikan jälkeen tulevia alkioita siirretään listan loppuun päin:

```
>>> lista.insert(3, 22)
>>> print lista
[2, 4, 6, 22, -5, 10, 12, 14, 16, -3]
```

# Muita metodeja listan käsittelyyn

- Metodi `remove` poistaa ensimmäisen alkion, jolla on parametrina annettu arvo:

```
>>> print lista
[2, 4, 6, 22, -5, 10, 12, 14, 16, -3]
>>> lista.remove(22)
>>> print lista
[2, 4, 6, -5, 10, 12, 14, 16, -3]
```

- Metodi `sort` järjestää listan:

```
>>> lista.sort()
>>> print lista
[-5, -3, 2, 4, 6, 10, 12, 14, 16]
```

- Metodi `reverse` kääntää listan järjestyksen päinvastaiseksi.

```
>>> lista.reverse()
>>> print lista
[16, 14, 12, 10, 6, 4, 2, -3, -5]
```

# Listojen yhdistäminen

- Kaksi listaa voidaan yhdistää käyttämällä operaattoria +:

```
>>> lista1 = [1, 2, 3]
>>> lista2 = [4, 5, 6]
>>> yhteislista = lista1 + lista2
>>> print yhteislista
[1, 2, 3, 4, 5, 6]
```

# Merkkijonot

- Merkkijonojen avulla ohjelmassa voi esittää tekstitietoa, esim. nimiä, osoitteita ja erilaisia tunnuksia.
- Merkkijonon tyyppi Pythonissa on `str`.
- Yksittäisiä merkkejä varten ei ole omaa tyyppiä, vaan ne ovat yhden merkin pituisia merkkijonoja.
- Merkkijono esitetään yksin- tai kaksinkertaisten lainausmerkkien avulla.

```
mjono = 'appelsiini'
mjono = "appelsiini"
```

# Merkkijonojen käsittely

- Merkkijonoja voidaan käsitellä monessa tapauksessa samalla tavalla kuin listoja, esim.

```
>>> sana = "sitruuna"
>>> print sana[3]
r
```

- Olennainen ero: merkkijonon sisältöä ei voi muuttaa sen jälkeen, kun merkkijono on luotu, esim.

```
>>> sana[3] = 'a'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

# Merkkijonon läpikäynti

- Merkkijonon merkit voi käydä läpi for-käskyn avulla samalla tavalla kuin listan alkiot:

```
>>> mjono = "matti"
>>> for merkki in mjono:
...     print merkki
...
m
a
t
t
i
```

# Alimerkkijonot

- Merkkijonosta voi ottaa myös alimerkkijonoja samaan tapaan kuin listoista alilistoja:

```
>>> mjono = "appelsiini"
>>> print mjono[2:5]
pel
>>> print mjono [:4]
appe
>>> print mjono [:-1]
appelsiin
```

# Merkkijonon pituus ja merkin esiintymisen tutkiminen

- Merkkijonon pituuden saa selville funktiolla `len`.

```
>>> mjono = "appelsiini"
>>> print len(mjono)
10
```

- Operaattorin `in` avulla voi tutkia, esiintyykö merkki merkkijonossa.

```
>>> print "i" in mjono
True
```

- Operaattorin `in` avulla voi myös tutkia, esiintyykö pitempi merkkijono osana toista.

```
>>> print "else" in mjono
False
>>> print "elsi" in mjono
True
```