Ohjelmoinnin peruskurssi Y1

CS-A1111

11.10.2023

Oppimistavoitteet: tämän luennon jälkeen

- Tunnet lisää erilaisia tapoja käyttää listoja Python-ohjelmissa.
- Tunnet erilaisia tapoja käsitellä merkkijonoja. Osaat esimerkiksi vertailla merkkijonojen sisältöjä.
- Osaat tehdä ohjelmia, jotka käsittelevät tehokkaasti tilanteita, joissa tarvitaan avain–arvo-pareja (esim. opiskelijarekisteri, puhelinluettelo)
- Voit luennon aikana lähettää kysymyksiä ja kommentteja sivulla `http://presemo.aalto.fi/y1syksy2023`

Alilistat

- Listasta voi ottaa helposti alilistoja (alkuperäisen listan osia):

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

Tulostus

```
[6, 8, 10]
```

- Ensimmäinen tai viimeinen indeksi voidaan myös jättää pois:

```
print(lista[:5])
```

Tulostus

```
[2, 4, 6, 8, 10]
```

```
print(lista[5:])
```

Tulostus

```
[12, 14, 16]
```

Alkoita listan lopusta

- Negatiiviset indeksit tarkoittavat alkioita listan lopusta lähtien:

```
lista = [2, 4, 6, 8, 10, 12, 14, 16]
print(lista[-1])
```

Tulostus

```
16
```

```
print(lista[:-1])
```

Tulostus

```
[2, 4, 6, 8, 10, 12, 14]
```

Listan järjestäminen ja kääntäminen

- Metodi `sort` järjestää listan:

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

Tulostus

```
[-5, -3, 2, 4, 6, 10, 14, 16]
```

- Metodi `reverse` kääntää listan järjestyksen päinvastaiseksi.

```
lista.reverse()
print(lista)
```

Tulostus

```
[16, 14, 10, 6, 4, 2, -3, -5]
```

Listasta järjestetty kopio

- *Funktio* `sorted` tekee listasta kopion ja järjestää tämän kopion. Alkuperäinen lista jää entiselleen:

```
lista1 = [4, 6, 10, 16, 14, 2, -3, -5]
lista2 = sorted(lista1)
print("Uusi lista:", lista2)
print("Alkuperainen lista:", lista1)
```

Tulostus

```
Uusi lista: [-5, -3, 2, 4, 6, 10, 14, 16]
Alkuperainen lista: [4, 6, 10, 16, 14, 2, -3, -5]
```

Listojen yhdistäminen

- Kaksi listaa voidaan yhdistää käyttämällä operaattoria +:

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

Tulostus

```
[1, 2, 3, 4, 5, 6]
```

Välitehtävä 1

- Mitä seuraava ohjelma tulostaa? Vastaa sivulla http://presemo.aalto.fi/y1syksy2023

```
lista1 = [1, 2, 3]
lista2 = [4, 5, 6]
lista3 = lista2 + lista1
lista4 = lista3[1:4]
lista4.sort()
print(lista4[1])
```

Merkkijonot

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

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

- Useammalle riville jatkuva merkkijono kirjoitetaan kolmen lainausmerkin sisään.

```
pitkajono = """Tama merkkijono sisaltaa
useamman kuin yhden
rivin"""
```

Merkkijonojen käsittely

- Merkkijonoja voidaan käsitellä monessa tapauksessa samalla tavalla kuin listoja:

```
sana = "sitruuna"
print(sana[3])
```

Tulostus

```
r
```

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

```
sana[3] = 'a'
```

Tulostus

```
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)
```

Tulostus

```
m
a
t
t
i
```

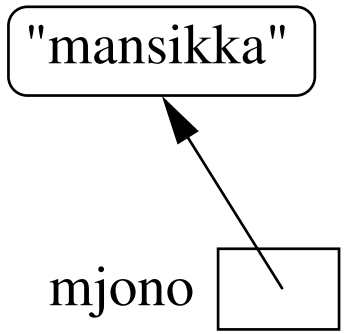
Sijoitus merkkijonomuuttujaan

- Merkkijonon sisältöä ei voi muuttaa sen jälkeen, kun merkkijono on luotu.
- Voidaan kuitenkin tehdä uusi merkkijono ja sijoittaa se arvoksi vanhalle muuttujalle:

```
mjono = "mansikka"
print(mjono)
```

Tulostus

```
mansikka
```



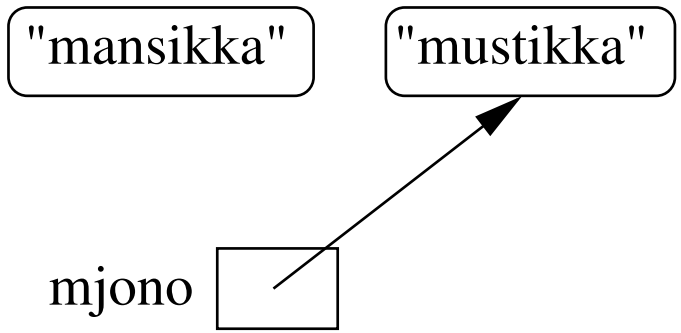
Sijoitus merkkijonomuuttujaan

- Merkkijonon sisältöä ei voi muuttaa sen jälkeen, kun merkkijono on luotu.
- Voidaan kuitenkin tehdä uusi merkkijono ja sijoittaa se arvoksi vanhalle muuttujalle:

```
mjono = "mansikka"
print(mjono)
```

Tulostus

```
mansikka
```




```
mjono = "mustikka"
print(mjono)
```

Tulostus

```
mustikka
```

Uusi merkkijono vanhan avulla

- Metodi `upper` luo uuden merkkijonon, jossa kaikki vanhan merkkijonon pienet kirjaimet on muutettu isoiksi:

```
mjono = "Mustikka"
mjono = mjono.upper()
print(mjono)
```

Tulostus

```
MUSTIKKA
```

- Metodi `lower` luo uuden merkkijonon, jossa kaikki vanhan merkkijonon isot kirjaimet on muutettu pieniksi:

```
mjono = "MUSTIKKA"
mjono = mjono.lower()
print(mjono)
```

Tulostus

```
mustikka
```

Merkkijonojen liittämiset

- Useampi merkkijono voidaan liittää peräkkäin +-operaattorilla.

```
etunimi = "Matti"
sukunimi = "Virtanen"
kokonimi = etunimi + " " + sukunimi
print(kokonimi)
```

Tulostus

```
Matti Virtanen
```

Jos merkkijonoon halutaan liittää muuntyyppisten muuttujien arvoja, pitää ensin suorittaa tyypinmuunnos `str`-operaattorilla.

```
tunteja = 50
tuntip = 12.5
rivi = str(tunteja) + " h * " + str(tuntip) + " eur / h"
print(rivi)
```

Tulostus

```
50 h * 12.5 eur / h
```

Merkkijonojen monistaminen

- Operaattorin * avulla voidaan tehdä merkkijono, joka sisältää pienemmän merkkijonon monta kertaa.

```
merkit = "*!*"
rivi = 5 * merkit
print(rivi)
```

Tulostus

```
*!**!**!**!**!*
```

- Rivinvaihdon saa mukaan erikoismerkillä `"\n"`:

```
rivit = 3 * (rivi + "\n")
print(rivit)
```

Tulostus

```
*!**!**!**!**!*
*!**!**!**!**!*
*!**!**!**!**!*
```

Erikoismerkkejä

- Merkkijonoihin on mahdollista liittää erikoismerkkejä (engl. escape characters), jotka aiheuttavat tulostuksessa esimerkiksi rivinvaihdon tai kursorin siirron seuraavaan tabulointikohtaan.
- Tärkeimpiä erikoismerkkejä:

`\n`	rivinvaihto
`\t`	tabulaattori
`\'`	yksinkertainen lainausmerkki
`\"`	kaksinkertainen lainausmerkki
`\\`	yksi kenoviiva

Tyhjien merkkien poisto merkkijonon alusta ja lopusta

- Halutaan poistaa merkkijonon alusta ja lopusta ns. tyhjät merkit.
- Tehdään metodin `strip` avulla:

```
teksti = "    \tjotain kirjoitusta     "
riisuttu_teksti = teksti.strip()
print(f"Alkuperainen: *{teksti:s}*")
print(f"Riisuttu: *{riisuttu_teksti:s}*")
```

Tulostus

```
Alkuperainen: *    jotain kirjoitusta     *
Riisuttu: *jotain kirjoitusta*
```

- Jos tyhjät merkit halutaan poistaan vain merkkijonon alusta tai lopusta, käytetään metodia `lstrip` tai `rstrip`.

Merkkijonon jakaminen

- Halutaan jakaa merkkijono osiin jonkun merkin (esimerkiksi välilyönnin) kohdalta.
- Merkkijono voidaan jakaa metodilla `split`. Se palauttaa listan, joka sisältää jaetun merkkijonon eri osat.
- Oletusarvoisesti `split`-metodi jakaa merkkijonon välilyönnin kohdalta, mutta metodin parametrilla voidaan määrätä merkki, jonka kohdasta jako tehdään.
- Jaossa käytetty merkki ei tule mukaan mihinkään osaan.

Merkkijonon jakaminen, esimerkkejä

```
teksti = "Pitka teksti, joka sisaltaa monta sanaa."
osat = teksti.split()
print(osat)
```

Tulostus

```
['Pitka', 'teksti,', 'joka', 'sisaltaa', 'monta', 'sanaa.']
```

```
sanarivi = "kirja=book"
kaannokset = sanarivi.split("=")
print(kaannokset)
```

Tulostus

```
['kirja', 'book']
```

Välitehtävä 2

- Tarkastellaan ohjelmaa

```
def main():
    rivi = input("Syota tulosrivi.\n")
    osat = rivi.split("/")
    yhteispisteet = osat[1] + osat[2]
    print("Pisteet yhteensa:", yhteispisteet)

main()
```

- Oletetaan, että käyttäjä syöttää rivin

  `Teemu Teekkari/25/15`

- Mitä ohjelma tulostaa? Vastaa sivulla
  http://presemo.aalto.fi/y1syksy2023

Merkkijonojen vertailu

- Merkkijonojen sisältöjä voi verrata toisiinsa vertailuoperaattoreilla ==, !=, <=, >=, < ja >.
- Tällöin verrataan merkkejä keskenään merkkijonojen alusta lähtien.
- Järjestyksen määrää kirjainten arvo käytetyssä merkkikoodausjärjestelmässä – mitä lukuarvoa kukin kirjain vastaa.
- Käytännössä koodit noudattavat muuten aakkosjärjestystä, mutta isot kirjaimet ovat ennen pieniä ja skandinaaviset aakkoset eivät ole keskenään oikeassa järjestyksessä.

Esimerkkejä merkkijonojen vertailuista

```
nimi1 = "matti"
nimi2 = "teppo"
print(nimi1 == nimi2)
```

Tulostus

```
False
```

```
print(nimi1 < nimi2)
```

Tulostus

```
True
```

```
nimi3 = "Teppo"
print(nimi2 == nimi3)
```

Tulostus

```
False
```

Esimerkkejä merkkijonojen vertailuista, jatkoa

```
nimi2 = "teppo"
nimi3 = "Teppo"
print(nimi3 < nimi2)
```

Tulostus

```
True
```

```
nimi1 = "matti"
nimi4 = "matilda"
print(nimi1 < nimi4)
```

Tulostus

```
False
```

Sanakirja

- Halutaan tallentaan *avain–arvo-pareja*. Myöhemmin rakenteesta pitää voida etsiä tiettyyn avaimeen liittyvää arvoa.
- Esimerkkejä: puhelinluettelo, opiskelijarekisteri, yrityksen asiakasrekisteri, autorekisteri.
- Halutaan, että lisäys, poisto ja arvon muuttaminen ovat helppoja.
- Yksinkertainen ratkaisu: käytetään listaa, jonka alkoina on avain–arvo-pareja. Ongelmia:
  - Hidas haku
  - Jos avaimia pidetään järjestyksessä, haku nopeutuu, mutta lisäys ja poisto hankaloituvat.
- Pythonissa on valmis rakenne, *sanakirja* (engl. dictionary), jossa sekä haut, lisäykset että poistot pystytään tekemään tehokkaasti.

Sanakirjan luonti ja käyttö

- Tyhjän sanakirjan voi luoda aaltosulkujen avulla:

```
puh_luettelo = {}
```

- Sanakirjaa luodessa voi samalla jo antaa siihen liitettäviä avain–arvo-pareja:

```
puhelinluettelo = {"Teekkari Teemu" : "050-12345",
    "Fyysikko Tiina" : "045-234567", "Kemisti Kalle" :
    "040-765432"}
```

- Haluttuun avaimeen liittyvän arvon saa selville ilmauksella `sanakirja[avain]`, esimerkiksi

```
print(puhelinluettelo["Fyysikko Tiina"])
```

Tulostus

```
045-234567
```

Sanakirja: avaimen haku ja olemassaolo

- Ohjelma voi kaatua, jos haettua avainta ei löydy sanakirjasta:

```
print(puhelinluettelo["Virtanen Maija"])
```

Tulostus

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'Virtanen Maija'
```

- Operaattorin `in` avulla voi tutkia, onko haettava avain sanakirjassa:

```
nimi = "Virtanen Maija"
if nimi in puhelinluettelo:
    print(puhelinluettelo[nimi])
else:
    print("Nimea ei loydy luettelosta")
```

Tulostus

```
Nimea ei loydy luettelosta
```

Sanakirja: avainten lisääminen ja arvon muuttaminen

- Sijoituskäskyn avulla sanakirjaan voi lisätä uusia avain–arvo-pareja ja muuttaa sanakirjassa jo oleviin avaimiin liittyviä arvoja.

```
puhelinluettelo["Rakentaja Niina"] = "0400-123"
puhelinluettelo["Kemisti Kalle"] = "041-56789"
print(puhelinluettelo)
```

Tulostus

```
{'Teekkari Teemu': '050-12345', 'Fyysikko Tiina':
'045-234567', 'Kemisti Kalle': '041-56789',
'Rakentaja Niina': '0400-123'}
```

Sanakirja: avainten läpikäynti

- Sanakirjan avaimet voi käydä läpi `for`-käskyn avulla.

```
for nimi in puhelinluettelo:
    print(nimi)
```

Tulostus

```
Teekkari Teemu
Fyysikko Tiina
Kemisti Kalle
Rakentaja Niina
```

```
for nimi in puhelinluettelo:
    print(f"{nimi:16s} {puhelinluettelo[nimi]:12s}")
```

Tulostus

```
Teekkari Teemu   050-12345
Fyysikko Tiina   045-234567
Kemisti Kalle    041-56789
Rakentaja Niina  0400-123
```

Sanakirja: avaimet järjestyksessä

- Pythonin versiosta 3.7 lähtien avain–arvo-parit ovat sanakirjassa samassa järjestyksessä kuin avaimet on lisätty sanakirjaan.
- Jos on mahdollista, että ohjelmaa käytetään tätä vanhemman Python-version tulkilla, ei voi luottaa minkäänlaiseen järjestykseen sanakirjassa.
- Jos taas halutaan, että avaimia käsitellään "aakkosjärjestyksessä", voidaan käyttää apuna funktiona `sorted`.

Sanakirja: avaimet järjestyksessä (jatkuu)

- Funktio `sorted` palauttaa **listan**, joka sisältää parametrina annetun sanakirjan avaimet järjestyksessä. Itse sanakirja jää ennalleen.

```
nimet_jarjestyksessa = sorted(puhelinluettelo)
for nimi in nimet_jarjestyksessa:
    print(f"{nimi:16s} { puhelinluettelo[nimi]:12s}")
```

Tulostus

```
Fyysikko Tiina    045-234567
Kemisti Kalle     041-56789
Rakentaja Niina   0400-123
Teekkari Teemu    050-12345
```

Sanakirja: avaimen poistaminen

- Sanakirjasta voi poistaa avaimen ja siihen liittyvän arvon `del`-operaattorilla:

```
del puhelinluettelo["Kemisti Kalle"]
print(puhelinluettelo)
```

Tulostus

```
{'Teekkari Teemu': '050-12345', 'Fyysikko Tiina':
'045-234567', 'Rakentaja Niina': '0400-123'}
```

Esimerkki: arvon päivittäminen sanakirjassa vanhan arvon avulla

- Luettu arvo lisätään sanakirjassa samaan avaimeen aikaisemmin kuuluneeseen arvoon.
- Jos avainta ei ole sanakirjassa, siihen lisätään uusi avain–arvo-pari.

```
myyntimaarat = {"Ketku" : 5000.0, "Hukkanen" : 8000.0,
                "Ovela" : 4500.0}
edustaja = input("Kenen tietoihin uusi kauppa lisataan?\n")
maara = float(input("Anna lisattava summa.\n"))
if edustaja in myyntimaarat:
    myyntimaarat[edustaja] = myyntimaarat[edustaja] + maara
else:
    myyntimaarat[edustaja] = maara
print(myyntimaarat)
```