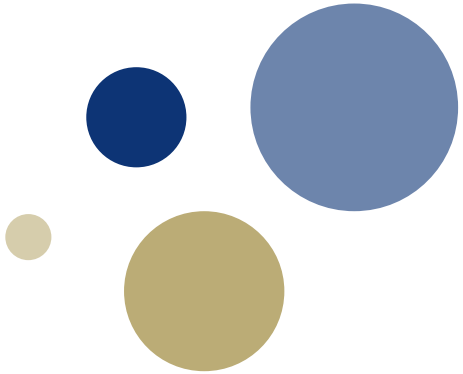
NTNU

Kunnskap for en bedre verden



Python: Oppslagslister (dictionaries) og mengder

3. utgave: Kapittel 9

TDT4110 IT Grunnkurs

Professor Guttorm Sindre

Læringsmål og pensum

- Mål
  - Forstå prinsippene for, og kunne bruke i praksis
    - Mengder (sets)
    - Oppslagslister (dictionary)
  - Vite forskjell på disse og lister, tupler, strenger
  - Vite forskjell på tekstfiler og binærfiler
  - Forstå, og kunne bruke, serialisering av objekter
- Pensum
  - Starting out with Python, Chapter 9:
    - Dictionaries and Sets

Kapittel 9.2

Mengder (Sets)

Mengder (Eng.: sets)

- Datastruktur for matematiske mengder
- Hvert element i en mengde er unikt
  - Umulig å legge inn dubletter
- Mengder er ikke ordnet
  - kan ikke anta noen bestemt rekkefølge
- De fleste funksjoner for lister kan brukes for mengder
- Mengder har flere spesifikke operasjoner, bl.a.:
  - *intersection* = no. snitt
  - *union*
  - Ulike typer mengdedifferanser

Opprette og endre på mengder

- En mengde opprettes ved å bruke funksjonen set( )

  A = set([5,3,3,12])

  B = set([True, 'ost', 23.2, 92, False])

  C = set("aaabcd") # gir { 'a' , 'b ', 'c' , 'd' }

- Eller ved direkte angivelse av innhold:

  D = {9,4,1,3}

- Legge til og fjerne elementer fra en mengde:

  A.add(9) # Legg til ett element

  A.update([3,4,2]) # Legg til flere elementer

  B.remove(92)

Iterasjon og medlemsskap

- For-løkke kan iterere over en mengde
  - Som for lister, tupler, filer, strenger

```
for x in mengde:
    print(x)          # Skriver ut elementer under hverandre
```

- Undersøke om verdi tilhører mengde: in , not in
  - Tilsvarer ∈ og ∉ i matematisk notasjon

```
if x in A:
    print("Verdien" , x , "er i mengden A")
```

Mengdeoperasjoner på A og B

- Union av to mengder:

  C = A.union(B) # C = A U B

- Snitt av to mengder:

  C = A.intersection(B) # C = A ∩ B

- mengdedifferanse:

  C = A.difference(B) # C = A \ B

- Symmetrisk mengdedifferanse:

  C = A.symmetric_difference(B) # C = (A U B) \ (A ∩ B)

- Sjekke delmengde eller supermengde:

  A.issubset(B) # True hvis A ⊂ B, ellers False

  A.issuperset(B) # True hvis A ⊃ B, ellers False

Oppgave: mengder

- Start med koden mengde-uferdig.py
  - Hvis du ikke har den, holder det å gjenta input-setningene
- Lag et program som
  - Bruker mengdeoperasjoner
  - Finner mulige mistenkte for en tenkt forbrytelse
- Videre hint om hva som må gjøres står i koden…

Oppgave: mengde-uferdig.py

Løsning: mengde-ferdig.py

Eksempel: Syv fjell e.l.

- I mange byer fins arrangementer hvor man skal besøke et visst antall fjelltopper eller andre punkter av interesse innen en viss tidsperiode
- Anta at vi har fått inn navnelister fra hvert av disse punktene. Vi ønsker nå å generere to ting:
  1. En oversikt over alle deltagere, dvs. alle som fins på minst en av de innleverte listene
  2. En oversikt over de ivrigste deltagerne, dvs. de som har klart å besøke alle punktene (og dermed fins på alle liste)

Oppgave: fjell-uferdig.py

Løsning: fjell-ferdig.py

Kapittel 9.1

Oppslagslister (Dictionary)

Oppslagsliste (Dictionary)

- Lagrer en samling av data.
- Minner litt om lister men har klare forskjeller:
  - Defineres ved { } (ALT+SHIFT 8/9 på Mac)
  - Indeks kan være hva som helst (ikke bare tall)
    - F.eks strenger, heltall, flyttall, boolske verdier
    - Indeks kalles nøkkel (key)
  - Brukes når vi vil gjøre oppslag på en verdi
    - for å finne andre assosierte verdier

```
A = {}                          # Tom dictionary
A['Kari'] = 92925492            # Oppretter et element
tlf={'Jo' : 73540000 , 'Per' : 92542312 , 'Else' : 54239212}
print(tlf['Per'])               # Skriver ut verdien 92542312
```

Bruk av in og not in

- Sjekke om elementer finnes
  - som lister, mengder etc.

```
tlf={'Jo' : 73540000, 'Per' : 92542312, 'Else' : 54239212}
navn = input(’Hvem vil du vite nummeret til? ’)
if navn in tlf:
    print(tlf[navn])
else:
    print(’Har ingen opplysninger om ’ + navn)
```

Operasjoner for *dictionaries*

Operasjon	Forklaring
len(d)	Antall elementer i d
d[k]	Verdi til element i d med nøkkel k
d[k] = v	Sett element k til verdi v
del d[k]	Slett element k i d
d.clear()	Fjerne alle element i d
d.copy()	Lag kopi av d
~~d.has_key(k)~~	Fjernet i Python 3. Må bruke **in** i stedet.

Operasjon	Forklaring
d.items()	Returnerer liste av (nøkkel,verdi) par
d.keys()	Returnerer liste av nøkler i d
d.values()	Returnerer liste av verdier i d
d.get(k)	Samme som d[k]
d.get(k,v)	Returnerer d[k] hvis k er gyldig, ellers v

Ulike datatyper i dictionaries

- Verdier i dictionary kan også være lister
- Eks. lagre tlf, adresse og betalingsinfo i en dictionary:

```
db = {}                          # Oppretter dictionary
db['Jo']=[90503020, 'Logata 4, 7000 Trondheim' , True]
db['Ann']=[50201020, 'Strandg. 2, 5000 Bergen' , False]
print(db['Jo'])                  # skriver ut lista med data for Jo
```

- Fordel med dictionary vs. f.eks liste av lister:
  - –Kan slå opp direkte på meningsfylt indeks
    - Mens indeks i liste ofte er intetsigende (kun rekkefølge)
  - –Dictionary: mindre strev med å lete etter riktig element
    - så lenge *nøkkelen* er det vi leter etter

for-løkke og *dictionary*

- Bruker man for-løkke på en *dictionary*, løper man igjennom nøklene til *dictionary*:

```
d = {‘navn’ : ’Per’ , ’alder’ : 28, ’IQ’ : 129}
for x in d:                    # går igjennom nøkler i d
        print(x)               # skriver ut nøklene navn, alder, IQ
```

- For å sikre gjennomgang av nøkler alfabetisk:

```
for x in sorted(d.keys()):     # sorterer nøkler
        print(x)
```

Eksempel på ”lur” bruk av nøkkel

- Norges Scrabbleforbund (NSF) har ei tekstfil
  - drøyt 622.000 ord som er lovlig å skrive i Scrabble (på norsk)
  - Alfabetisk sortert, greit for å sjekke om et ord er lovlig eller ikke
- Vi vil lage et program som finner anagrammer
  - Bruker kan skrive inn en tekststreng
  - Finne alle ord i NSF-lista som bruker de samme bokstavene
  - Men i fri rekkefølge (ETTER, ERTET, RETTE…)
- Vil unngå å søke gjennom hele fila / lista for hver streng
  - Mål: Kunne slå opp direkte om et ord har anagrammer eller ikke
  - Da kan oppslagsliste (dictionary) være en lur datastruktur
  - Men hva bør vi bruke som nøkkel?
    - Nå er ord for ord, alfabetisk sortert, ikke lenger det gunstigste…

Ide til løsning:

- Regne om hvert ord til et tall
  - Identiske bokstaver → samme tall
  - Forskjell i antall eller type bokstav → annet tall
  - HVORDAN??
- Kan omregne hvert ord til et produkt av primtall
  - 2, 3, 5, 7, 11, 13, 17,…., 109 (primtall nr 29)
  - Hver bokstav assosieres med ett av disse tallene
  - Det blir store tall for lange ord
    - men enklere enn å søke gjennom hele lista hver gang
  - For å redusere størrelsen på tallene noe
    - bruk de laveste tallene for de vanligste bokstavene
    - E=2, T=3, R=5, …
    - Både ET og TE vil få nøkkel 6 (2*3)
    - Både ERT, TER og TRE vil få nøkkel 30 (2*3*5)

Ide til løsning (forts.)

anagram1.py

- Bygger så en dictionary ved å
  - Opprette tom dictionary
  - Gjenta for hvert ord i ordlistefila:
    - Les inn ordet
    - Regne ut nøkkel for ordet (produkt av primtall)
    - Putte ordet inn i dictionary på denne nøkkelen
      - HVIS flere ord på samme nøkkel: append() i liste…
      - ELLERS (første ord på denne nøkkelen): lage liste med ordet
  - Returnere full dictionary
- Når brukeren skriver inn en tekst og vil finne anagram:
  - Regne ut nøkkelverdi for brukerens tekst
  - HVIS denne nøkkelen fins i dictionary:
    - Returnere lista av ord (anagrammer) som tilhører nøkkelen
  - ELLERS:
    - Returnere tom liste (ingen anagrammer funnet)

Kapittel 9.3

Serialisering av objekter

Serialisering av objekter

- Konverterer et objekt til en strøm av bytes
  - Gir en persistent variabel
  - Data lagres til fil, kan senere leses tilbake
  - Verdiene overlever programslutt, at maskin slås av, etc.
  - I Python: biblioteket *pickle* har funksjoner for dette
- Fordeler med binærfiler (vs. tekstfil)
  - Får dataene direkte inn i ønsket datastruktur
  - Slipper konvertering til / fra strenger
  - Vanligvis mindre plasskrevende
- Ulemper vs. tekstfil
  - Fila er ikke lesbar for mennesker
  - Applikasjonsavhengig
    - må ha spesifikk kjennskap til filformatet for å kunne bruke fila

Lagre dictionary til disk

- Bruk biblioteket pickle
  - metoden pickle.dump( )
- Må åpne fila som binærfil og ikke som tekst,
  - det er mer enn tekst som lagres (struktur):

```
db={ 'Jo' : [10, 'Skogata 3'], 'Per' :[20, 'Heigata 2' ]}
import pickle                          # Importerer modulen
f = open( 'database.dat' , 'wb' )      # b=binary
pickle.dump(db,f)                      # Dumper db til disk
f.close()                              # Lukker fila
```

Laste inn dictionary fra disk

- Metoden pickle.load()
- Husk å åpne fila som binærfil og ikke tekst.

```
import pickle
f = open( 'datafil.dat' , 'rb' )    # r=read, b=binary
data = pickle.load(f)               # Laster inn dict fra disk
f.close()
```

Anagram-finneren, V2

- Serialisering kan redusere ventetid for brukeren
- Dele opp i tre filer:
  - En modul for felles kode: anagram2.py
  - Et ”backstage”-program: anagram2-backstage.py
    - Leser ordlista fra Scrabbleforbundet
    - Bygger opp dictionary og dumper den på fil
  - Et program for brukeren: anagram2-bruker.py
    - Leser inn dictionary fra fil
    - Spør brukeren om tekster og finner anagrammer
- Brukeren trenger kun kjøre brukerprogrammet
    - Backstage-programmet kjøres sjeldnere
      - F.eks kun hver gang det kommer ny versjon av ordlista
    - Brukeren slipper mye venting i starten ☺

Kahoot

- 10 spørsmål om
  - Mengder, oppslagslister og serialisering
  - Basert på det som forelesningen har dekket

Oppsummering

- Mengder (sets) er nyttige for å gjøre operasjoner som
  - Snitt, union og mengdedifferanse
  - Mengder defineres ved å bruke funksjonen **set(liste)**
- Oppslagsliste (dictionary) ligner på lister, men…
  - Man kan bruke hva som helst som nøkkel (indeks).
  - Kan opprettes tom: **A = {}**
  - Eller ved nøkkel + data: **A = {'navn': 'Petter Ole'}**
  - Trenger ikke indeks som er i rekkefølge
  - Kan serialiseres / dumpes til binærfil