

Python: Mengder og Dictionaries

3. utgave: Kapittel 9

TDT4110 IT Grunnkurs
Professor Guttorm Sindre

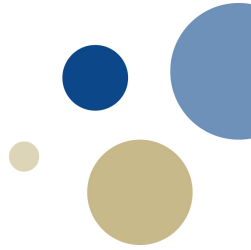
Læringsmål og pensum



- Mål
 - Forstå prinsippene for, og kunne bruke i praksis
 - Mengder (sets)
 - "Ordbøker" (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

Foregående to uker vs. denne

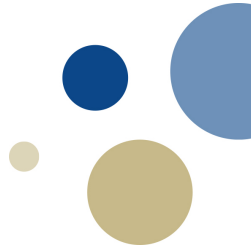
- Sammensatte variable: lagre data i primærminnet
 - Lister; tupler; strenger
 - [1, 3, 3, 5] ['a','e','i','o','u','y'] ; (1, 3, 3, 5) ('a','e','i','o','u','y') ; '1335', 'aeiouy'
 - Rekkefølge på elementene. Lister er muterbare, tupler og strenger **ikke**
 - Mengder, dictionaries
 - {1, 3, 5} {'a','e','i','o','u','y'}
 - {'a': '-.-', 'b': '-...-', 'c': '-.-.'} {'010101 23456': ['Per', 'Olsen', 'Trondheim']}
 - Muterbare, ingen fast rekkefølge, ingen duplikater
 - FORDEL: Raskere oppslag, raskere å legge til eller fjerne element
 - ULEMPE: Større plassforbruk enn lister, tupler, strenger
- Filer: lagre data permanent i sekundærminne
 - Tekstfiler: leses og skrives til/fra strengvariable
 - FORDEL: Bredt anvendelig, lett å lese av mange slags apper
 - ULEMPE: Litt treg lesing, stor plassbruk, tidsbruk til konvertering
 - Binærfiler: kan leses/skrives til/fra andre variable (f.eks. dictionary)
 - FORDEL: Mer kompakt lagring, raskere lesing, slipper konvertering
 - ULEMPE: Kan bare leses av visse program



Kapittel 9.2

Mengder (Sets)

Mengder (Eng.: sets)



- Datastruktur for matematiske mengder
- Unike element, ingen dubletter
- Ikke ordnet, dvs. ingen bestemt rekkefølge
- De fleste funksjoner for lister kan brukes for mengder
- Mengder har flere spesifikke operasjoner, bl.a.:
 - *union*
 - *intersection* = no. snitt
 - 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}
```

```
# NB: D = {} gir en tom dictionary. Tom mengde: D = set()
```

- 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)              # fjerner tallet 92
```

Iterasjon og medlemskap



- 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`
 - Tilsvarende \in og \notin 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 ∪ 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 ∪ B) \ (A ∩ B)`

- Sjekke delmengde eller supermengde:

`A.issubset(B)` `# True hvis $A \subset B$, ellers False`

`A.issuperset(B)` `# True hvis $A \supset B$, ellers False`

Oppgave

LETTERE:

Start med koden

`mengde_uferdig.py`

Fullfør koden ved å skrive ferdig de to funksjonene `lag_mengde()` og `finn_mistenkte()`.

Personer skal komme fram som mulige mistenkte hvis de hadde våpen og motiv, men ikke alibi

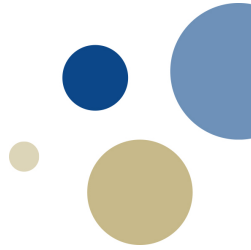
Løsning: `mengde-ferdig.py`

VANSKELIGERE:

Ser for oss et trimløp hvor de besøker flest mulig fjelltopper på en dag. Får inn ei liste av lister, hvor hver indre liste er de som har vært på en bestemt topp. Start med koden `fjell_uferdig.py`

Fullfør koden ved å skrive ferdig de to funksjonene, bruk mengder for å finne hvem som har deltatt (minst en topp) og hvem som har vært på alle toppene.

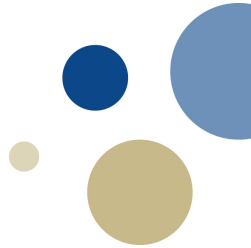
Løsning: `fjell-ferdig.py`



Kapittel 9.1

“Ordbøker” (Dictionary)

Ordbok (Dictionary)



- Lagrer en samling av data.
- Defineres ved { } (ALT+SHIFT 8/9 på Mac)
- Nøkkelverdi kan være hva som helst (ikke bare tall)
 - F.eks strenger, heltall, flyttall, boolske verdier
 - Indeks kalles nøkkel (key)
 - Gjøre oppslag på en verdi, finne assosierte verdier

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

Operasjoner for *dictionaries*

in for å sjekke om nøkkelverdi fins

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

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

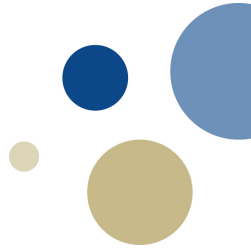
Ulike datatyper i dictionaries

- 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 nøkkelverdi
 - 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*:

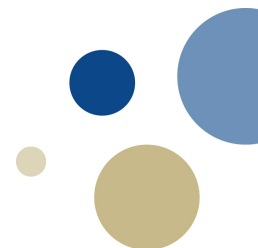
```
d = {'navn' : 'Per' , 'alder' : 28, 'IQ' : 129}
```

```
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)
```

Hva bruke som nøkkel?



- Kommer an på anvendelsen, hva vi vil slå opp på
 - Eks.: Persondata
 - Personnummer.
 - Fordel: unikt. Ulempe: husker det ikke, bør ikke spres for mye
 - Telefonnummer... navn...
 - Eks.: liste med ord
 - Oversettelse norsk-engelsk: { 'god': 'good', 'stor': 'big' }
 - Gradbøying: { 'god': ['bedre', 'best'], 'stor': ['større', 'størst'] }
 - Rimkatalog: { 'erte': ['berte', 'erte', 'fjerte', 'hjerte', 'smerte', 'terte'] }
 - Anagrammer: finne ord med samme bokstaver uavh. av rekkefølge
 - { ??? : ['ad', 'da'], ??? : ['ert', 'ter', 'tre'], ??? : ['etter', 'rette', 'treet'] ... }
 - Hva bruke som nøkkel her?

Ide til nøkkel for anagrammer:

- 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 ($2 \cdot 3$)
 - Både ERT, TER og TRE vil få nøkkel 30 ($2 \cdot 3 \cdot 5$)

Oppgave

- Laging av dictionary

LETTERE:

Start med koden

`liste_og_dict_V0.py`

Skriv funk. `lag_dictionary()`

på samme måte som

`lag_2D_liste()` bare at den

legger ordene i en

dictionary, f.eks

```
{'god': ['bedre', 'best'],  
  'stor': ['større', 'størst']}
```

heller enn 2d liste

```
[ ['god', 'bedre', 'best'],  
  ['stor', 'større', 'størst'] ]
```

Løsning: `liste_og_dict_V1.py`

VANSKELIGERE:

Start med koden

`anagram_V0.py`

Skriv funksjonen

`string_2_key()` som gir

nøkkeltall for et ord,

og `list_2_dic()` som går

igjennom hele lista av ord

og lager en dictionary hvor

alle anagrammer er

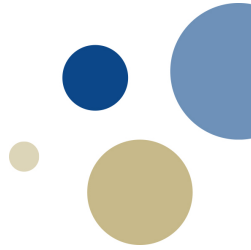
plassert ved samme

nøkkel. **NB: Plasser fila**

`nsf2016.txt` på samme

katalog som koden

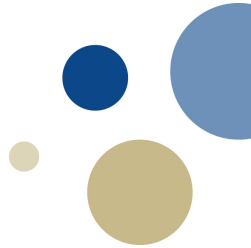
Løsning: `anagram_V1.py`



Kapittel 9.3

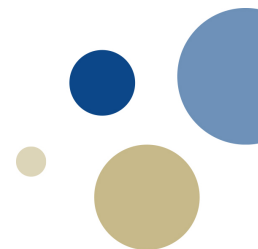
Serialisering av objekter

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

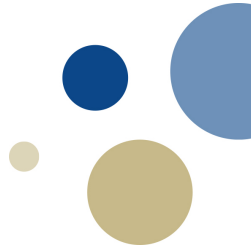
Lagre dictionary til disk



- 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: `anagram_v2_felles.py`
 - Et admin-program `anagram_v2_admin.py`
 - Leser ordlista fra Scrabbleforbundet fra tekstfil
 - Bygger opp dictionary og dumper den på binærfil
 - Et program for brukeren: `anagram_v2_user.py`
 - Leser inn dictionary fra binærfil
 - Spør brukeren om tekster og finner anagrammer
- Brukeren trenger kun kjøre brukerprogrammet
 - Admin-programmet kjøres sjeldnere
 - F.eks kun hver gang det kommer ny versjon av ordlista
 - Brukeren slipper mye venting i starten 😊

Oppsummering

- Mengder (sets) er nyttige for å gjøre operasjoner som
 - Snitt, union og mengdedifferanse
 - Mengder defineres ved å bruke funksjonen **set()**
- Ordbok (dictionary)
 - Kan bruke hva som helst som nøkkel, eks.
 - **D = {'Norge': ['Oslo', 'Bergen', 'Trondheim']}**
 - Trenger ikke indeks som er i rekkefølge
 - Kan serialiseres / dumpes til binærfil