

TDT4110 Informasjonsteknologi grunnkurs:

Tema: Dictionaries og sets (mengder)

Utgave 3: Kap. 9

Terje Rydland - IDI/NTNU

Læringsmål og pensum

• Mål

- Forstå prinsippene for, og kunne bruke i praksis
 - Mengder (sets)
 - Ordbøker (dictionary)
- Vite forskjell på disse og lister, tuller og strenger
- Vite forskjell på tekstfiler og binærfiler
- Forstå og kunne bruke serialisering av objekter

• Pensum

- Starting out with Python, Chapter 9: Dictionaries and Sets

Datatyper

- Vi har hittil sett på:
 - Lister [1,3,5,7] ['a','e','i','o'] muterbare
 - Tupler (1,3,5,7) ('a','e','i','o') ikke muterbar
 - Strenger '1357' 'aeio' ikke muterbare
 - For alle gjelder: Rekkefølge er viktig, man kan ha duplikater
- Nå skal vi se på:
 - Mengder {1,3,5,7} {'a','e','i','o'} muterbar
 - Dictionaries {1 : 'a' , 2 : 'e' , 3 : 'i' , 4 : 'o' } muterbar
 - For disse gjelder: **ingen fast rekkefølge, ingen duplikater**
- Fordeler med mengder og dictionaries:
 - **Raskere** oppslag, raskere å oppdatere
- Ulemper med mengder og dictionaries
 - **Større plassforbruk** enn lister, tupler og strenger

Filer

- Vi har hittil sett på tekstfiler:
 - Leser og skriver til fil via en tekstvariabel
 - `f = open(filnavn,'r')`
 - `f.close()`
 - **Fordel:** Anvendelig, kan leses av mange forskjellige program
 - **Ulempe:** Treg lesing, tar plass, må konverteres, ikke alle data kan lagres som tekst
- Nå skal vi se på binærfiler:
 - Binærfiler kan brukes til data det ikke er naturlig å lagre som tekst
 - **Fordel**
 - Mer kompakt lagring, raskere lesing, slipper konvertering
 - **Ulempe**
 - Kan bare leses av noen program

Kapittel 9.2

Mengder (SETS)

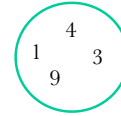
Datastruktur for mengde (sets)

- Python har innebygd en datastruktur for å håndtere **mengder** (sets på engelsk).
- **Hvert element i en mengde er unik**, så du vil ikke få lagt til et element som finnes fra før.
 - Bruk dette f.eks. til å finne unike ord i en tekst?
- Mengder er **ikke ordnet**: kan ikke anta rekkefølgen.
- De fleste funksjoner for lister kan brukes for mengder.
- Mengder har flere spesifikke operasjoner bla.:
 - intersection (snitt): Kun elementer representert i begge mengdene
 - union: Alle elementer fra begge mengdene (kun en av hver)

Opprette og endre på mengder

- En mengde kan **lages** ved å tilordne en serie elementer inne i {}

```
D = {9,4,1,3} # Lager en mengde
```



- En kan **konvertere lister og strenger** ved å bruke funksjonen **set(x)**

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

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

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

```
E = set() # gir en tom mengde
```

- 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 test av mengder

- På samme måte som for lister og filer, kan for-løkke brukes for å iterere over en mengde:

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

- Kan også undersøke om en verdi finnes eller ikke finnes i en mengde ved bruk av **in** eller **not in**:

```
if 1 in A:
    print("Verdien 1 er i mengden A")
if 5 not in A:
    print("Verdien 5 er ikke i mengden A")
```

Mengdeoperasjoner på A og B

- Union av to mengder:

`C = A.union(B)` # C blir union av A og B

- Snitt av to mengder:

`C = A.intersection(B)` # C blir snitt av A og B

- Forskjellen på to mengder:

`C = A.difference(B)` # Finnes i A, men ikke i B

- Symmetrisk forskjell på to mengder:

`C = A.symmetric_difference(B)` # Elementer som ikke deles av A og B

- Sjekke delmengde eller supermengde:

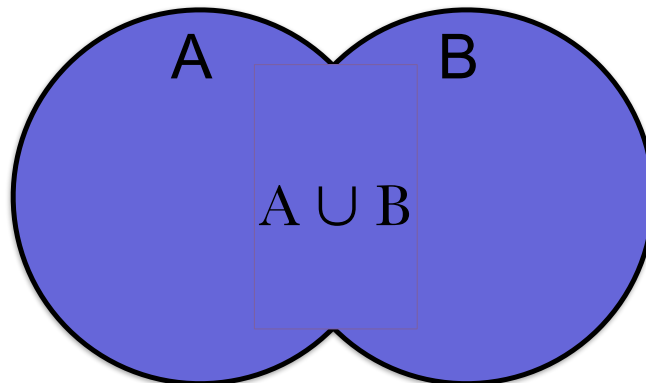
`A.issubset(B)` # Sjekker om A er delmengde av B

`A.issuperset(B)` # Sjekker om A er supermengde av B

Union av to mengder - Venn diagram

- $A \cup B$

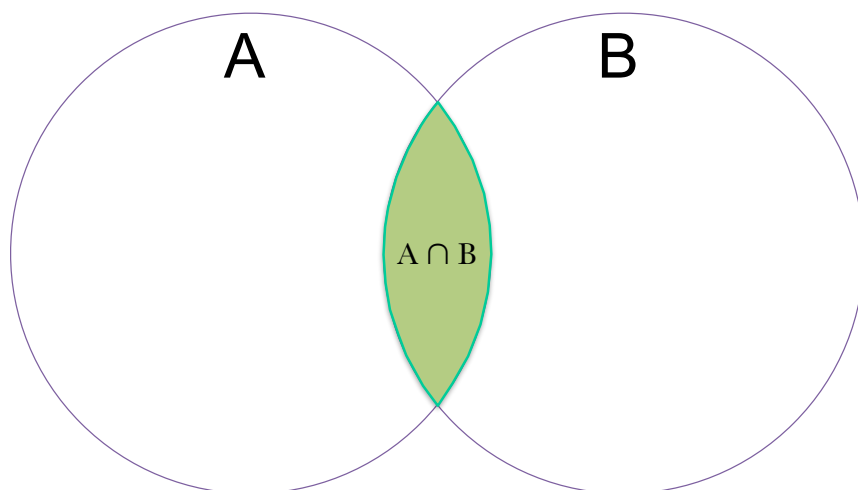
`C = A.union(B)` # C blir union av A og B



Snittet av to mengder - Venn diagram

• $A \cap B$

`C = A.intersection(B)` # C blir snitt av A og B

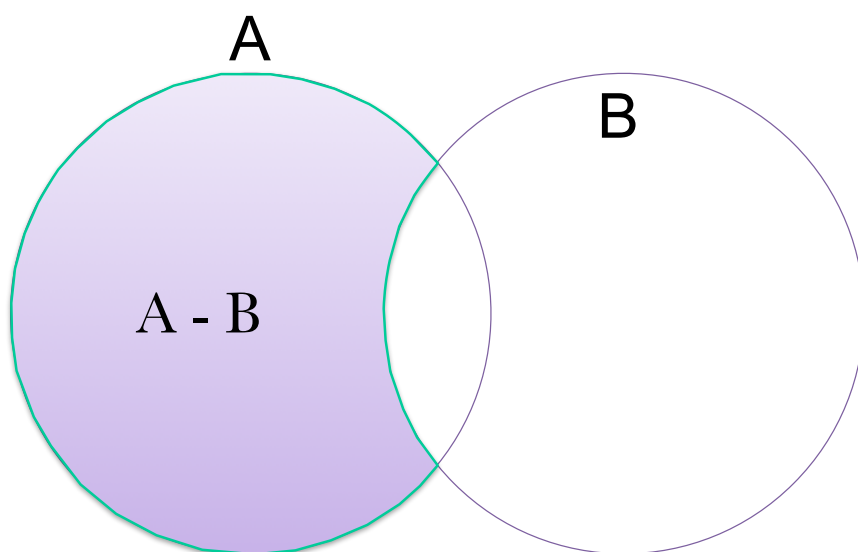


lotto.py

Forskjell på mengde A og B - Venn diagram

• $A - B$

`C = A.difference(B)` # Finnes i A, men ikke i B

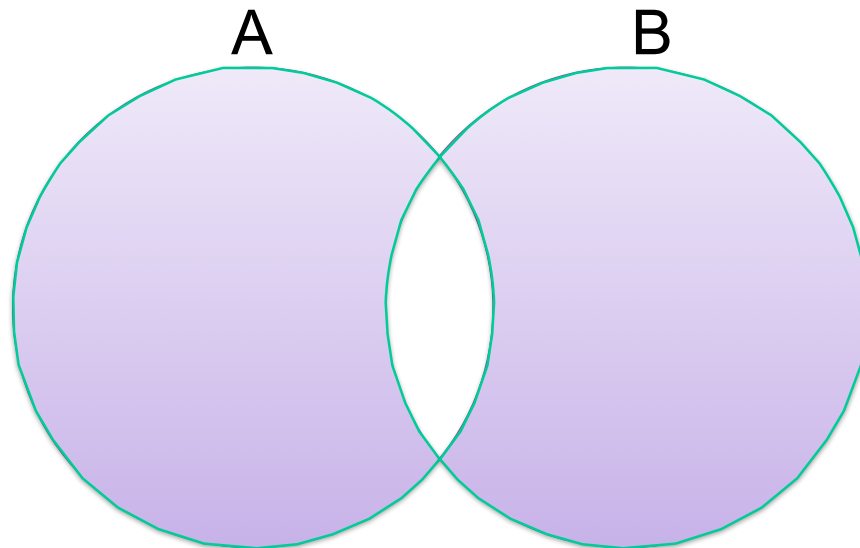


Symmetrisk forskjell på A og B

- Venn diagram

- $A \Delta B \quad ((A \cup B) - (A \cap B))$

`C = A.symmetric_difference(B)` # Elementer som ikke deles av A og B



Oppgave: mengder

- Skriv Python-koden for å gjøre følgende:
 - Spør bruker om å skrive inn to mengder ved hjelp av input, og lagre dette som mengdene A og B.
 - Skrive ut resultatet til skjerm av følgende:
 - Snitt av A og B (intersection)
 - Union av A og B (union)
 - Forskjell på A og B (difference)
 - Symmetrisk forskjell på A og B (symmetric_difference)
 - Test om A er delmengde av B (issubset)



`mengder.py`

Oppgave: mengder



```
*mengder.py - /Users/terjery/Library/Mobile Documents/com~apple~CloudDocs/ITGK 2016/Python/Keynote/Uke 44/Python/mengder...
A = set(eval(input("Skriv inn mengden A: ")))
B = set(eval(input("Skriv inn mengden B: ")))

print("Snitt av A og B er:",A.intersection(B))
print("Union av A og B er:",A.union(B))
print("Forskjell paa A og B er:",A.difference(B))
print("Symmetrisk forskjell paa A og B er:",A.symmetric_difference(B))
print("Sjekker om A er delmengde av B:",A.issubset(B))
```

Ln: 2 Col: 4

16

Datastruktur: *Dictionaries* Kap 9.1

- Dictionary er et objekt som lagrer en samling av data.
- Minner litt om lister men har klare forskjeller:
 - Defineres ved å bruke krøllparanteser { } (ALT+SHIFT 8/9 på Mac)
 - Kan bruke **hva som helst som nøkkel** (indeks) (ikke bare tall som i lister):
 - Tekststrenger, Heltall (men trenger ikke å være i rekkefølge), Flyttall, Sannhetsverdier (True eller False), En kombinasjon av de ovenfor

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

- {nøkkel₁ : verdi, nøkkel₂ : verdi, ..., nøkkel_n : verdi}

Bruk av operatorene **in** og **not in** i Dictionaries

- Man kan bruke **in** og **not in** i dictionaries for å sjekke om elementer finnes (nøkler!):

```
tlf={'Jo':73540000,'Per':92542312,'Else':54239212}
if ('Per' in tlf):
    print(tlf['Per'])
if ('Lars' not in tlf):
    print('Lars er ikke i dictionaryen')
```

Operasjoner for *dictionaries*

| Operasjon | Forklaring | Operasjon | Forklaring |
|---------------------------|--|-------------------------|---|
| <code>len(d)</code> | Antall elementer i d | <code>d.items()</code> | Returnerer liste av (nøkkel,verdi) par |
| <code>d[k]</code> | Verdi til element i d med nøkkel k | <code>d.keys()</code> | Returnerer liste av nøkler i d |
| <code>d[k] = v</code> | Sett element k til verdi v | <code>d.values()</code> | Returnerer liste av verdier i d |
| <code>del d[k]</code> | Slett element k i d | <code>d.get(k)</code> | Samme som <code>d[k]</code> |
| <code>d.clear()</code> | Fjern alle elementer i d | <code>d.get(k,v)</code> | Returnerer <code>d[k]</code> hvis k er gyldig, ellers v |
| <code>d.copy()</code> | Lag kopi av d | | |
| <code>d.has_key(k)</code> | Fjernet i Python 3. Må bruke in | | |

dictionary_metoder.py

dictionary_metoder.py

```

dictionary_metoder.py - /Users/terjery/Library/Mobile Documents/com~apple~CloudDocs/ITGK 2016/Python/Keyno...
d = {} # Oppretter et dictionary

print('Skriv inn key og verdi. Avslutt med tom streng.')

key = input('Key: ')

while(key!=''):
    value = input('Value: ')
    d[key] = value

    key = input('Key: ')

print('len(d):', len(d))
print('d.items():', d.items())
print('d.keys():', d.keys())
print('d.values():', d.values())
print('d.get(\'ost\', -999):', d.get('ost', -999))

```

Ln: 1 Col: 0

Ulike datatyper i dictionaries

- Dictionaries kan brukes til å representere sammensatt informasjon ved hjelp av lister.
- Lister gjør det mulig å knytte en nøkkel til ulike typer data. Eks. lagre telefonnummer, adresse og om vedkommende har betalt i en dictionary:

```

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

```

[90503020, 'Logata 4, 7000 Trd', True]

- 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:

```
d = {'navn': 'Per', 'alder': 28, 'IQ': 29}
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_dictionary.py

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

```
for_dictionary.py - /Users/terjery/Library/Mobile Documents/com~apple~CloudDocs/ITGK 2016/Python/Keynote/Uk...
d = {'navn': 'Ole', 'alder': 42, 'weight': 45} #Oppretter dict
print(d)
print('\nKeys non-sorted:\n')
for x in d:
    print(x, d[x])
print('\nKeys sorted:\n')
for x in sorted(d.keys()): # sorterer keys
    print(x, d[x])
Ln: 1 Col: 0
```

Nyttig bruk av *dictionary*

- Etersom vi kan bruke hva som helst som nøkkel i en *dictionary*, egner den seg svært godt til f.eks. å telle forekomster i strenger, i lister eller lignende.

```
f = {} # Oppretter dictionary for å telle forekomster
for element in liste: # sjekker elementer i ei liste
    f[element] = f.get(element, 0) + 1
# Legg til +1 i dictionary med nøkkelelement
```

telling_dictionary.py

```
telling_dictionary.py - /Users/terjery/Library/Mobile Documents/com~apple~CloudDocs/ITGK 2016...
print('Skriv inn setninger og avslutt med slutt')
freq = {}
line = input()
while(line != 'slutt'):
    words = line.split()
    for word in words:
        freq[word] = freq.get(word, 0) + 1
    line = input()
for word in freq:
    print(word, '\ter skrevet', freq[word], 'ganger')
Ln: 1 Col: 0
```

Hva skal vi 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
 - Oversettelsenorsk-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?

Idé til nøkkel for anagrammer

- Regne om hvert ord til et tall
 - Identiske bokstaver → samme tall
 - Forskjellig 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)

Kapittel 9.3

Serialisering av objekter

Serialisering av objekter

Kap 9.3

- Serialisering av objekter er prosessen å **konvertere et objekt til en strøm av bytes** som kan lagres til fil, som senere kan lastes inn igjen.
 - F.eks. for lagring av **dictionary** eller **set**
- En persistent variabel, betyr at verdier overlever selv om man avslutter Python og avslutter datamaskinen.
- Python har biblioteket pickle som gjør det mulig å lagre og laste dictionary til/fra disk.

- 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

- For å lagre en dictionary til disk kan **biblioteket pickle** brukes sammen med **metoden dump**.
- Må åpne en fil som binærfil og ikke som tekst, ettersom det er mer enn tekst som lagres (struktur):

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

lagre_dictionary.py

lagre_dictionary.py

```

pickle.py - /Users/terjery/Library/Mobile Documents/com~apple~CloudDocs/ITGK 2016/Python/Eksempel på forel...
import pickle # Importerer pickle biblioteket

print('Skriv inn navn, telefon og adresse. \
Avslutt med ENTER når programmet ber om navn.')

navn = input('Oppgi navn: ')

tlf = {} # Oppretter en tom dictionary

while navn != '':
    telefon = input('Telefonnr: ')
    adresse = input('Adresse: ')
    tlf[navn] = [telefon,adresse]
    navn = input('Oppgi navn: ')

filnavn = input('Oppgi navn på fila du vil lagre data til: ')

f = open(filnavn,'wb') # Åpner fila for binær skriving
pickle.dump(tlf,f) # Konverterer til binært og
# dumper dictionary tlf til disk
f.close() # Stenger fila

```

Ln: 1 Col: 23

lagre_dictionary.py

```

pickle 2.py - /Users/terjery/Library/Mobile Documents/com~apple~CloudDocs/ITGK 2016/Python/Eksempler på forelesning/pic...
import pickle # Importerer pickle biblioteket

print('Skriv inn navn, telefon og adresse. \
Avslutt med ENTER når programmet ber om navn.')

tlf = {} # Oppretter en tom dictionary

while True:
    navn = input('Oppgi navn: ')
    if navn == '':
        break # Avslutt while-løkke og hopp ut
    telefon = input('Telefonnr: ')
    adresse = input('Adresse: ')
    tlf[navn] = [telefon,adresse]

filnavn = input('Oppgi navn på fila du vil lagre data til: ')

f = open(filnavn,'wb') # Åpner fila for binær skriving
pickle.dump(tlf,f) # Konverterer til binært og
# dumper dictionary tlf til disk
f.close() # Stenger fila

```

Ln: 3 Col: 0

Laste inn dictionary fra disk

- Laste inn dictionary fra disk gjøres ved hjelp av load metoden i pickle-biblioteket.
- Husk at 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()

```

```

*laste_dictionary.py - /Users/terjery/Library/Mobile Documents/com~apple~CloudDocs/ITGK 2016/Python/Keynote/Uke 44...
import pickle # Importerer pickle bibliotek

filnavn = input("Skriv inn navn på datafil: ")

f = open(filnavn,"rb") # Åpner for lesing og binærfil
database = pickle.load(f) # Laste inn dictionary fra disk
f.close() # Stenger fila

for item in database:
    print(item,":",database[item])

```

laste_dictionary.py

Ln: 12 Col: 0

Oppsummering

- Sets er nyttige for å gjøre operasjoner på mengder:
 - intersection og union
 - Sets defineres ved å bruke funksjonen `set(liste)`
- Dictionary ligner på lister men man kan bruke hva som helst som nøkkel (indeks).
 - Opprettes ved å bruke `A = {}` eller `A = {'navn': 'Petter Ole'}`
 - En fordel med dictionary er at man ikke trenger en indeks som er i rekkefølge.
 - Det finnes flere kommandoer som man kan bruke på dictionary.
 - Dictionary oppfører seg som en database.