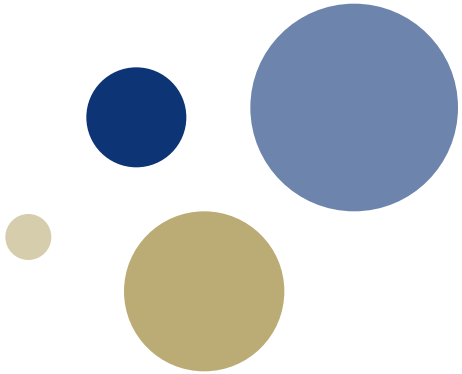
NTNU

Kunnskap for en bedre verden



Python: Strenger
(Pluss mer om lister og tupler)

3. utgave: Kapittel 8 (og 7)

TDT4110 IT Grunnkurs

Professor Guttorm Sindre

Læringsmål og pensum

- Mål
  - Forstå
    - Hva strenger er
    - Grunnleggende operasjoner på strenger
    - Indeksering av tegn i strenger, inkl. *slicing*
  - Kjenne til noen vanlige funksjoner og metoder på strenger
  - Kunne løse praktiske programmeringsproblemer med strenger

- Pensum
  - Starting out with Python:

  Chapter 8 More About Strings (3rd edition)

Tekststrenger (string)

- HVORFOR trenger vi dette?
  - Informasjon: tekst spiller en sentral rolle
  - Veldig mange nyttige programmer behandler tekst
    - Tekstbehandling, weblesere, søkemotorer
    - Journalistikk, publisering
    - Saksbehandlingssystemer
    - Teknisk informasjon, brukerveiledninger, …
    - Oversettingsprogrammer
- Tekststrenger i Python
  - En av de elementære datatype
  - Men også en sekvens av enkelttegn
    - Som tuppel: ikke muterbar

Aksessere deler av strenger

- Få tak i enkelttegn: bruke indeks (kap 8.1)

```
tekst = 'Dette er en test'
tekst[0]          # Gir 'D'
tekst[14]         # Gir 's'
tekst[-1]         # Gir 't', siste tegn (nest siste: -2, osv.)
```

- Få tak i større deler av en streng: slicing (kap 8.2)
  - Syntaks: string[start : end : step]

```
tekst = 'Dette er en test'
tekst[0:3]          # Gir 'Det'     ~ tekst[:3]
tekst[12:16]        # Gir 'test'    ~ tekst[12:]
tekst[::2]          # Gir 'Dtee nts'    (annenhvert tegn)
```

for-løkker gjennom strenger (8.1)

- Se på ett og ett tegn
- Som for lister: to muligheter
  - Aksessere tegnene direkte: enklest
    - Som her, skal bare skrive teksten loddrett
  - Aksess via indeks
    - Tungvint her, trengs hvis posisjon i strengen er viktig
      - F.eks finne hvor i strengen det står en dobbel konsonant

```
s = 'Dette er en test'
# Aksessere tegnene direkte
for t in s:        # t blir 'D', så 'e', så 't'…
    print(t)
```

```
s = 'Dette er en test'
# Aksessere tegnene via indeks
for i in range(len(s)):    # i blir 0, så 1, 2, …, 14
    print(s[i])            # da blir s[i] 'D', 'e', 't', osv.
```

Forskjell på funksjoner og metoder

- Gjelder både strenger, lister, tupler…
- Funksjoner er frittstående
  - Strengen (eller lista, tuplet, …) som funksjonen skal jobbe på, må gis inn som argument når funksjonen kalles, f.eks.

```
len(s)      # gir lengde av strengen s
list(s)     # gir ei liste av enkelttegnene i strengen
int(s)      # gir et heltall fra strengen (hvis mulig)
```

- Metoder ”eies” av strengobjektet (eller listeobjektet…)
  - Strengen selv gis IKKE inn som argument i ( )
  - Står i stedet foran metodekallet med dot-notasjon, f.eks.

```
s.upper( )          # returnerer tilsv. streng med bare store bokstaver
s.replace('å', 'aa') # returnerer tilsv. streng, alle å byttet med aa
liste.sort( )       # ber liste om å sortere seg selv (fins ikke for
                    # strenger, som er immutable)
```

Strengoperasjoner, forts.

- Sette sammen flere strenger til en (konkatenering)
  - Bruk operatoren + , f.eks.

```
navn = fornavn + etternavn
```

  - Sette et tall inn i en streng, bruk str():

```
respons = 'Du har ' + str(saldo) + 'kroner på kontoen'
```

  - Sette sammen liste av strenger: metoden **join()**

```
'-'.join(['Berkåk', 'Å', 'Meldal']) # gir 'Berkåk-Å-Meldal'
```

- Splitte en streng i flere strenger, metoden **split()**
  - Gir ei liste av delstrenger (unntatt skilletegnet)
  - Blank er default skilletegn, kan gi annet som argument

```
t = 'Dette er en test'
L = t.split()              # Gir: ['Dette', 'er', 'en', 'test']
u = '1;2;4;6;4;4;3'
V = u.split(';')           # Gir ['1', '2', '4', '6', '4', '4', '3']
```

Eksempel:

- Vi vil lage et program som gjør dette:

```
Beskriv deg selv med ett eller flere adjektiv.
Hvis flere, ha mellomrom mellom dem.
Gi inn adjektiv: snill flink dum ekkel
Din oppblåste dust!
Du tror du er snill, flink, dum og ekkel.
Hah! Jeg er mye snillere, flinkere, dummere og eklere.
```

Adjektivene skal tastes inn av brukeren, resten skrives av programmet

- Hvordan tenke her ?

Eksempel:

- Vi vil lage et program som følger:

```
Beskriv deg selv med ett eller flere adjektiv.
Hvis flere, ha mellomrom mellom dem.
Gi inn adjektiv: snill flink dum ekkel
Din oppblåste dust!
Du tror du er snill, flink, dum og ekkel.
Hah! Jeg er mye snillere, flinkere, dummere og eklere.
```

Adjektivene skal tastes inn av brukeren, resten skrives av programmet

- Hvordan tenke her ? (A): Overordnet algoritme…

  # 1: les inn adjektivene som en streng fra bruker

  # 2: splitt til ei liste av adjektiv, f.eks [’snill’, ’flink’, ’dum’, ekkel’]

  FORDI: vi må se enkeltadjektivene for å gradbøye

  # 3: lag ei tilsvarende liste av adjektiv i komparativ

  # 4: skriv den dissende teksten på skjermen

Eksempel:

- Vi vil lage et program som følger:

```
Beskriv deg selv med ett eller flere adjektiv.
Hvis flere, ha mellomrom mellom dem.
Gi inn adjektiv: snill flink dum ekkel
Din oppblåste dust!
Du tror du er snill, flink, dum og ekkel.
Hah! Jeg er mye snillere, flinkere, dummere og eklere.
```

Adjektivene skal tastes inn av brukeren, resten skrives av programmet

- Hvordan tenke her ? (B): Overordnet funksjonsinndeling

main()

- les_inn_adj()
  - # 1: les inn adjektivene som en streng fra bruker m input()
  - # 2: splitt til ei liste av adjektiv, f.eks [’snill’, ’flink’, ’dum’, ekkel’]
    FORDI: #2 er enkel, løses på ei kodelinje med .split()
- komp_liste()
  - # 3: lag ei tilsvarende liste av adjektiv i komparativ
- print_diss()
  - # 4: skriv den dissende teksten på skjermen

Kode: dissing_bm_V0.py dissing_nyn_V0.py

Eksempel:

- Vi vil lage et program som følger:

```
Beskriv deg selv med ett eller flere adjektiv.
Hvis flere, ha mellomrom mellom dem.
Gi inn adjektiv: snill flink dum ekkel
Din oppblåste dust!
Du tror du er snill, flink, dum og ekkel.
Hah! Jeg er mye snillere, flinkere, dummere og eklere.
```

Adjektivene skal tastes inn av brukeren, resten skrives av programmet

- Hvordan tenke her ? (C): Selve gradbøyinga

main()

↳ komp_liste()

Gå i for-løkke gjennom lista av adjektiv

Finn komparativ form for ett og ett adjektiv

Legg komparativ i ny liste

Returner lista når løkka er slutt

Mer komplekst. Best å lage egen funksjon, komparativ() som bøyer ett adj.
Lag hjelpefunksjoner for behov som opptrer flere steder:

- telle antall stavelser
- sjekke om bokstav er vokal

Regler for komparativ form (BM)

- Normalt: legge til -ere — snill – snillere
- -m dobles etter vokal — dum – dummere
  - men: kvalm - kvalmere
- Lange ord (>= 3 stavelser), mer… — desperat – mer desperat
- -et og >1 stavelse, mer… — lettet – mer lettet
  - men: het - hetere
- -el, -en, -er og >1 stavelse
  - Fjern e'en nest bakerst, legg til -ere — enkel-enklere, rusten-rustnere
    - men: pen-penere
  - Fjern dessuten en konsonant hvis dobbel: — vissen-visnere, vakker-vakrere
- Noen adjektiv har uregelrett bøying og må bare huskes:
  - god-bedre, stor-større, liten-mindre, korrupt-mer korrupt, …

Reglar for komparativ form (Nyn)

- Normalt: leggje til -are — snill – snillare
- -m doblast etter vokal — dum – dummare
  men: kvalm - kvalmare
- Lange ord (>= 3 stavingar), meir… — desperat – meir desperat
- -a, -ut og >1 staving, meir… — letta – meir letta, pysut – meir pysut
- -el, -en, -er og >1 staving
  - Fjern e'en nest sist, legg til -are — enkel-enklare, rusten-rustnare
    men: pen-penare
  - Fjern dessutan ein konsonant om dobbel: — vissen-visnare, vakker-vakrare

- Somme adjektiv har uregelrett bøying, desse må vi berre hugse:
  - god-betre, stor-større, liten-mindre, korrupt-meir korrupt, …

Oppgave: Fullfør komparativ( )

- Ta utgangspunkt i koden
  - dissing_bm_V1.py eller
  - dissing_nyn_V1.py

LETTERE:
(a) Skriv kode så komparativ( ) virker i det helt vanlige tilfellet, f.eks snill-snillere / snillare
(b) Utvid koden så den også takler dobling av m etter vokal, f.eks dum-dummere / dummare

MIDDELS:
(a) LETTERE + Utvid koden så den takler uregelrette adjektiv ved å bruke de globale variablene
(b) Utvid koden så den også takler adjektiv som slutter på -el, -en, -er (men uten sjekk på stavelser)

VANSKELIGERE
(a) Legg til en hjelpefunksjon for å telle antall stavelser i et ord
(b) Benytt hjelpefunksjonen til å utvide koden til å takle korrekt de reglene som er avhengig av antall stavelser

Fasit på dissing_bm_V2.py / dissing_nyn_V2.py

Testing, søking og manipulering av strenger

Kapittel 8.3

Strengoperasjoner (forts.)

- Sjekke om streng inneholder tegn / delstreng:
  - operatoren **in**
    - Generelt format: **streng1** **in** **streng2**
      - Kan f.eks brukes som betingelse i if- og while-setninger
    - Fins også en motsatt operator not in
- Finne hvor i en streng et tegn er: metoden index()

```
streng.index(tegn)          # finner første fra venstre
streng.rindex(tegn)         # første fra høyre (bakfra)
```

Strengmetoder

- Strenger i Python har mange ferdige metoder
  - Generelt format: mystring.method(arguments)
  - Testemetoder (**Table 8-1** i boka)
    - Betingelser for hele strenger
    - Returnerer True hvis betingelsen er sann ellers False.
  - Modifiseringsmetoder: (**Table 8-2** i boka)
    - Kopier av strengene hvor noe kan være endret
  - Søk- og erstatt-metoder (**Table 8-3** i boka)
    - Leter etter tegn / delstrenger i strenger
- Blir for oppramsende å forelese disse tabellene
- Sett deg ned i interaktiv modus og prøv deg fram!

Oppsummering

Dette kapittelet dekket:

- Operasjoner på strenger, som
  - Metoder for å iterere gjennom strenger
  - Operatorer for repetisjon og konkatinering
  - Strenger som ikke-muterbare objekter
  - Å *slice* og teste strenger
  - Metoder for strenger
  - Å splitte opp strenger

Mer om lister

- Todimensjonale lister / matriser
  - Kan lages i Python som lister av lister
  - F.eks.

$$\begin{pmatrix} 2 & 4 & -6 & 9 \\ 3 & 1 & -2 & 1 \\ 0 & 0 & 1 & -3 \end{pmatrix}$$

```
M = [ [2,4,-6,9], [3,1,-2,1], [0,0,1,-3] ]
```

- For å gå gjennom elementer i 2D-liste: dobbel løkke

```
def vis_matrise(M):                  # vise M på skjermen rad- og kolonnevis
    for r in range(len(M)):          # ant. element i ytre liste, dvs. ant. rader
        for k in range(len(M[0])):   # ant. element i indre liste (kolonner)
            print(M[r][k], end = " ")  # printe ett tall, unngå linjeskift
        print( )                     # indre løkke slutt, linjeskift før neste rad
```

Lister, referanser, kopiering

- Listevariabel inneholder referanse til datasekvensen

  `liste2 = liste1` vil gjøre at begge peker på samme data i minnet

  Jfr. du kan gi Maler 1 en lapp med adr. til huset ditt, så gi Maler 2 en kopi av samme lapp

- Hvis du deretter endre innholdet i lista via den ene variabelen, vil dataene som den andre ser også bli endret (siden det er samme data)

  Jfr., hvis Maler 2 drar og lager rosa prikker på huset ditt, vil Maler 1 også finne et rosaprikket hus hvis han drar til samme adresse

```
>>> liste1 = [1, 2, 3, 4, 5]
>>> liste2 = liste1
>>> liste2
[1, 2, 3, 4, 5]
>>> liste2 = [6, 7, 8]
>>> liste1
[1, 2, 3, 4, 5]
>>> liste2
[6, 7, 8]
>>> # hele liste2 endret til nytt innhold, liste1 IKKE påvirket
>>> liste2 = liste1
>>> liste2[1] = -99 # endrer andre element i liste2
>>> liste2
[1, -99, 3, 4, 5]
>>> liste1
[1, -99, 3, 4, 5]
>>> # BEGGE lister ble påvirket av endringen!
>>> # FORDI: liste1 og liste2 er referanser til samme data i minnet
>>> # Hvis vi ønsker noe annet må vi ta kopi:
```

Lister, kopiering (forts.)

- Noen ganger ønsker vi en kopi av ei liste med separate data
  - F.eks kunne endre på kopien uten å ødelegge de originale dataene
- Ulike måter å få til dette på
  - Slicing, konkatenering med tom liste, eller funksjonen list( )
  - Bruke funksjonene copy.copy() eller copy.deepcopy( )

```
>>> # BEGGE lister ble påvirket av endringen!
>>> # FORDI: liste1 og liste2 er referanser til samme data i minnet
>>> # Hvis vi ønsker noe annet må vi ta kopi:
>>> liste1 = [1, 2, 3, 4, 5]
>>> liste2 = liste1[:] # tar en slice som dekker hele lista
>>> liste3 = [] + liste1 # konkatenerer med en tom liste
>>> liste4 = list(liste1) # bruker list(), lager ny liste m samme verdier
>>> liste2[0] = 2
>>> liste2
[2, 2, 3, 4, 5]
>>> liste3[0] = 3
>>> liste3
[3, 2, 3, 4, 5]
>>> liste4[0] = 4
>>> liste4
[4, 2, 3, 4, 5]
>>> liste1
[1, 2, 3, 4, 5]
>>> # Nå er liste1 IKKE påvirket!
>>> # Alle listene viser til separate data i minnet
```

2D-lister og kopiering

- For 2D (og flerdimensjonale lister) blir kopiering mer komplekst
  - Hvert element i ytre liste er igjen en referanse til ei indre liste
  - De enkle triksene med [:] , [] + … eller list( ) funker derfor ikke
    - Må eventuelt gå i løkke og gjøre dette for hver innerste liste
  - Enklere da: bruke copy.deepcopy( )

```
>>> M = [ [1, 2, 3], [4, 5, 6] ]
>>> M
[[1, 2, 3], [4, 5, 6]]
>>> X = M[:]
>>> X[0] = [9, 9, 9]
>>> X
[[9, 9, 9], [4, 5, 6]]
>>> M
[[1, 2, 3], [4, 5, 6]]
>>> # byttet hel rad i X, da ble M IKKE påvirket
>>> X[1][2] = -888
>>> X
[[9, 9, 9], [4, 5, -888]]
>>> M
[[1, 2, 3], [4, 5, -888]]
>>> # Endret ett enkelt element, M ble også endret!
>>> import copy
>>> Y = copy.deepcopy(M)
>>> Y[0][0] = 'ost'
>>> Y
[['ost', 2, 3], [4, 5, -888]]
>>> M
[[1, 2, 3], [4, 5, -888]]
>>> # Her ble M ikke påvirket
```

- Opprettelse av nullmatrise
  - For å lage en nullvektor:
    - V = [0] * 3 # f.eks
  - Funker dårlig for 2D og mer
    - M = [[0] * 3] * 2
    - Gir 2 rader med 3 kol. 0'er
    - Begge rader er samme data
  - Bruk heller:

```
M = [ [ 0 for k in range(3) ] for r in range(2) ]
```