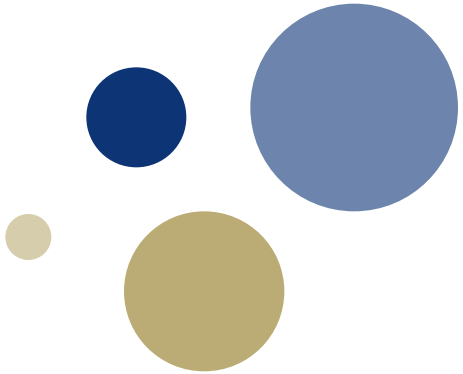
NTNU

Kunnskap for en bedre verden



Python: Lister og tupler

Gaddis: Kapittel 7

TDT4110 IT Grunnkurs

Professor Guttorm Sindre

Denne uka

Vi trenger å	Støttes av	
Hente data fra bruker	•Fra tastatur: input()	Andre former for input
Vise data til bruker	•Tekst til skjerm: print() m.m.	Andre former for output
Lagre data i minnet for bruk videre i programmet	•Variable, enkle datatyper: Heltall, flyttall, strenger, sannhetsverdier	•…sammensatte datatyper: Lister, tupler, mengder, dictionary, objekter / klasser
Lagre data permanent (og hente)	•Tekstfiler	•Binærfiler
Prosessere data	•Operatorer • = , +=… +, -, *… >, ==, …	•Innebygde funksjoner og metoder
Styre hvorvidt og hvor ofte programsetninger utføres •Valg •Repetisjoner	Kontrollstruktur •standard sekvens •if-setning •løkker (while, for)	Kontrollstruktur •Unntaksbehandling •Rekursjon
Gjøre programmet forståelig Bryte ned problemet i deler Oppnå fleksibilitet og gjenbrukbarhet	•Kommentarer •Funksjoner •Moduler	•Objektorientert design •Klasser og arv
Forstå hva vi har gjort feil	•Feilmeldinger	•Debugging

Læringsmål og pensum

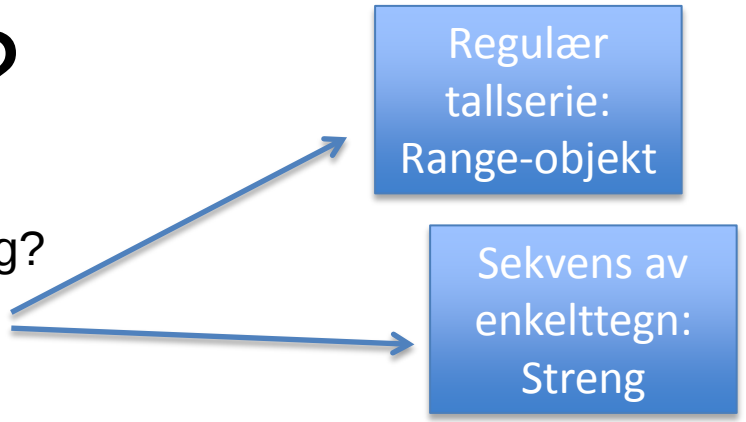
- Mål
  - Denne og senere uker: Vite om sammensatte datatyper i Python
  - …og velge riktig type ut fra problemet
  - Denne uka: Kunne løse programmeringsproblemer med
    - Lister og tupler
  - Mestre noen spesifikke Python-mekanismer for lister
    - Slicing, kopiering
    - Operatoren **in**
    - Vanlige innebygde liste-metoder og funksjoner
- Pensum
  - Starting out with Python, Chapter 7: Lists and Tuples

Tidligere lært

- Enkle datatyper: heltall, flyttall, strenger, lister
  - Hver variabel inneholder én verdi
    - `i = 5`
    - `navn = 'Nina'`
    - `medlem = True`
    - `score = 2.77`
- Datamaskiner skal behandle store mengder data
  - Kommer ikke langt med bare enkle datatyper
  - Tusen tall… lage tusen variable **tall1, tall2, …., tall1000**
  - Vanskelig å gå gjennom dette i løkke…
- Derfor: sammensatte datatyper
  - Hver variabel kan inneholde flere verdier
  - I vårt pensum: lister, tupler, mengder, dictionary

Sammensatte datatyper

Hva bruke når?



- Er rekkefølgen på elementer viktig?
  - JA: liste eller tuppel …. eller
  - NEI: mengde eller dictionary
- Kan dupliserte elementer forekomme?
  - JA: liste eller tuppel
  - NEI: mengde eller dictionary
- Trengs raske oppslag på verdi?
  - NEI: liste eller tuppel (oppslag ikke viktig, eller kun oppslag på indeks)
  - JA: mengde eller dictionary
    - Mengde: kun verdien - Dictionary: verdien er nøkkel til andre data
- Må data kunne endres ”in place”?
  - JA: liste, mengde, eller dictionary (de er muterbare - *”mutable”*)
  - NEI: tuppel (er ikke muterbar - *”immutable”* , à la strenger)

Hva bruke når? – i dag

- Er **rekkefølgen** på elementer viktig?
  - JA: liste eller tuppel
  - NEI: mengde eller dictionary
- Kan **dupliserte** elementer forekomme?
  - JA: liste eller tuppel
  - NEI: mengde eller dictionary
- Trengs raske **oppslag på verdi**?
  - JA: mengde eller dictionary
    - Mengde: kun verdien - Dictionary: verdien som nøkkel
  - NEI: liste eller tuppel (oppslag ikke viktig, eller kun på indeks)
- Vil vi **endre data** ”in place”?
  - JA: liste, (er muterbare - ”mutable”)
  - NEI: tuppel (er ”immutable”, a la strenger)

Sekvenser

Kapittel 7.1

Sekvenser

- *Sekvens:*
  - objekt med flere dataverdier i rekkefølge
- Python har flere sekvenstyper. Pensum:
  - Ikke-muterbare *(immutable)*:
    - Strenger
    - Range-objekter
    - Tupler:
      - Primært brukt for samlinger av heterogene data, f.eks.
      - pos = (x, y, z) # koordinater til en gjenstand i 3D
      - land_info = **(**'Norge', 47, 5.2, 'NOK', True**)**
  - Muterbare (*mutable*)
    - Lister:
      - Primært brukt for samlinger av homogene data, f.eks.
      - temp_siste_uke = **[**7.5, 8.9, 5.6, 10.1, 9.2, 5.5, 6.0**]**
      - terningkast = **[**1, 4, 3, 6, 2, 2, 2, 3, 6, 5**]**

Introduksjon til lister

Kapittel 7.2

Opprette lister

- Opprette liste i Python kan gjøres ved å…
  - ramse opp elementene:

    dager = [’Mandag’, ’Tirsdag’, ’Onsdag’, ’Torsdag’, ’Fredag’, ’Lørdag’, ’Søndag’]
  - lage en tom liste (sette inn elementer senere)

    kundeliste = [ ]

    kundeliste.append(’Petter’)
  - bruke ***** som repetisjonsoperator

    salg_pr_mnd = [0] * 12

    # lager lista [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
  - bruke funksjonen list( ) på et itererbart objekt som f.eks range()

    oddetall = list(range(1, 10, 2))

    # lager lista [1,3,5,7,9]
  - skrive en ”list comprehension”

    kvadrattall = [x * x for x in range(1,100)]

    # lager lista [1, 4, 9, 16, …, 9801, 10000]

Prosessering av lister

- Kan bruke / endre enkeltelementer ved indeksering
  - Første indeks er 0
  - Siste indeks er [n-1], der n er antall element i lista

  dager = [’Mandag’, ’Tirsdag’, ’Onsdag’, ’Torsdag’, ’Fredag’, ’Lørdag’, ’Søndag’]
  - Hente element, f.eks. **idag = dager[1]** **# gir ’Tirsdag’**
  - Endre element, f.eks. **dager[5] = ’Laurdag’** **# mulig fordi liste er muterbar**

- Kan også operere på hele lister, f.eks.
  - **print(dager)** # skriver ut hele lista
  - **len(dager)** # returnerer antall elementer i lista (her 7)
  - **ny_liste = dager + [0, 0, 0, 0]**

    # gir [’Mandag’, ’Tirsdag’, ’Onsdag’, ’Torsdag’, ’Fredag’, ’Lørdag’, ’Søndag’, 0, 0, 0, 0]

    # bruker + som konkateneringsoperator
  - **dager.sort( )** # sorterer elementer stigende (her alfabetisk)

For-løkker gjennom lister

- To ulike måter
  - Aksessere verdiene direkte
  - Aksessere via indeks
- Viktig å skjønne forskjellen!
  - Første eksempel: gjør samme på to ulike måter

```
liste = ['Per', 'Nina', 'Jo', 'Lucinda', 'Geir']
for navn in liste:
        print(navn)
# navn blir 'Per', så 'Nina', så 'Jo', ...
```

```
liste = ['Per', 'Nina', 'Jo', 'Lucinda', 'Geir']
for i in range(len(liste)):
        print(liste[i])
# i blir 0, 1, 2, 3, 4
# dermed blir liste[i] 'Per', 'Nina', 'Jo', ...
```

  - Andre eksempel: to ulike behov:

```
L = [5.32, 0.0, 5.67, 6.08, 0.0, 6.01]
lengste = L[0]
for x in L:   # x blir 5.32, 0.0, 5.67, ...
        if x > lengste:
                lengste = x
print('Lengste hopp:' , lengste)
```

Problemet krever bare verdiene.

```
L = [5.32, 0.0, 5.67, 6.08, 0.0, 6.01]
nr = 0 # indeks for lengste hopp
for i in range(len(L)): # range(6): i blir 0, 1, 2, 3, 4, 5
        if L[i] > L[nr]:
                nr = i
print('Lengste hopp:' , L[nr])
print(' kom i forsøk nr' , nr + 1)
```

Problemet krever også posisjon i lista

Oppgave: Hovedstadsquiz

LETTERE:
Start med koden **quiz_lett_v0.py**
Fullfør funksjonen still_spm() som går gjennom lista land i rekkefølge og spør hva som er hovedstad i hvert land. Du trenger **ikke** sjekke om brukeren svarer riktig. Returner antall spørsmål som ble stilt.

MIDDELS:
Start med koden **quiz_mid_v0.py**
Fullfør funksjonen still_spm() som går gjennom lista land i rekkefølge og spør hva som er hovedstad i hvert land. Sjekk mot lista fasit og si om brukeren svarer rett eller ikke, la funksjonen returnere antall rette.

VANSKELIGERE:
(a) Start med koden **quiz_van_v0.py**
Fullfør funksjonen still_spm() som spør i tilfeldig rekkefølge med random.choice( ) inntil det ikke er flere land å spørre om. Returner to lister: land hvor svaret var riktig og land hvor svaret var galt.

Løsninger ligger på tilsvarende filer med navn …_v1.py

Slicing (skiving) av lister

Kapittel 7.3

Å skive/slice lister

- Slice: et spenn av enheter som er tatt fra en sekvens
  - Slicing syntaks: **liste[start : slutt : inkrement]**
  - kopi av elementer fra opprinnelig liste
  - fra og med start, til (men ikke med) slutt
    - Hvis start ikke er spesifisert, brukes 0 som start
    - Hvis slutt ikke spesifiseres brukes len(liste) som slutt
    - Hvis inkrement ikke spesifisert brukes +1
  - negative indekser er relative til listens slutt

```
liste = [1,2,3,4,5,6]
x = liste[0:2]          # gir x = [1,2]
x = liste[3:-1]         # gir x = [4,5]
x = liste[1:6:2]        # gir x = [2,4,6]
```

Endring av lister ved hjelp av slice

- Kan også bruke slice-uttrykk til å endre innhold:

```
A = [1,2,3,4,5,6]       # Lager ei liste A med 6 element
A[:2] = [0,0]           # Endrer to første, gir A=[0,0,3,4,5,6]
A[-2:] = [9,9]          # Endrer to siste, gir A=[0,0,3,4,9,9]
A[0::2] = [5,5,5]       # Endrer annet hvert, gir A=[5,0,5,4,5,9]
A[-3:]=[ ]              # Bytter tre siste m. tom liste. Gir A=[5,0,5]
A[3:]=[4,5,6]           # Hekter på [4,5,6] etter siste. Gir A=[5,0,5,4,5,6]
A[3:3]=[1,1,1]          # Setter inn 1-ere v indeks 3. Gir A=[5,0,5,1,1,1,4,5,6]
```

- Innsetting på slutten, kan også bruke extend( )

```
A.extend([9,17])        # Gir A=[5,0,5,1,1,1,4,5,6,9,17]
```

- NB: Understreking _ i eksemplene kommer ikke i lista, er bare for å vise hvilke endringer som skjedde

Teste om et element fins i lista

Kapittel 7.4

in-operatoren

- Eksistens av element i liste kan testes med løkke:

```
funnet = False
i = 0
while (not funnet) and (i < len(liste)):
    if liste[i] == det_vi_leter_etter:
        funnet = True
    i+=1
```

- Enklere: **in**-operatoren… **if item in list:**
  - Vårt eksempel: **if det_vi_leter_etter in liste:**
    funnet = True
  - True hvis den er i listen, ellers False
- Men kan trenge løkke hvis vi skal gjøre noe mer
  - F.eks. vite posisjon til elementet, telle antall treff, fjerne duplikatelementer, eller lignende

Metoder og funksjoner for lister

Kapittel 7.5

Flere nyttige listeoperasjoner

Metoder: liste.metodenavn(), Funksjoner: funknavn(liste), Operator

- append(item) - innsetting til slutt
- extend(liste) – utvidelse til slutt
- insert(index, item) - inn v index
- index(verdi)
  - Indeks for 1. forekomst av verdien
- sort() – sorterer i stigende rkflg
- reverse() – snur lista
- remove(verdi)
  - Fjerner 1. forekomst av verdien
- len() – lengde av lista
- max(), min() - størst, minst
  - Hvis tekst: alfabetisk
- reversed() – returnerer liste i omvendt rekkefølge
- del liste[i]
  - Fjerner element m indeks i

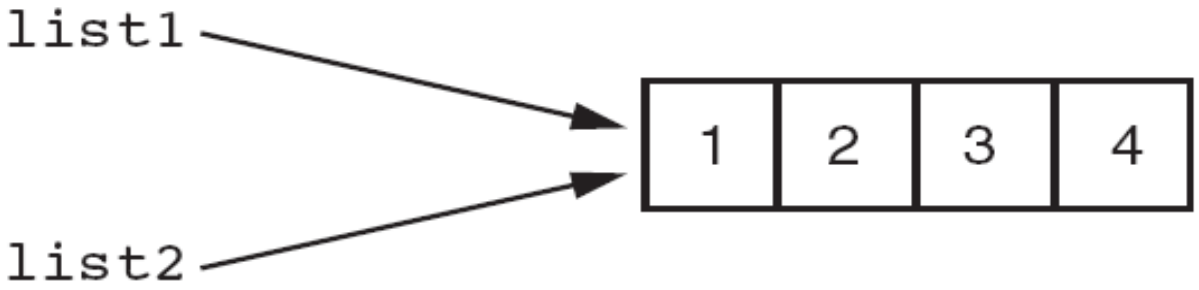
```
A=[1,2,3]
A.append(5) eller
A.extend([5])      # Gir A=[1,2,3,5]
A.insert(2,9)      # Gir A=[1,2,9,3,5]
n = A.index(9)     # Gir n = 3
# gir feilmelding hvis ikke funnet
A.sort( )      # gir A=[1,2,3,5,9]
A.reverse( )  # gir A=[9,5,3,2,1]
A.remove(2)  # gir A=[9,5,3,1]
T = len(A)     # gir T = 4
m = max(A)  # gir m = 9
B = reversed(A)  # gir B = [1,3,5,9]
                 # men A er uendret
del  A[2]      # gir A=[9,5,1]
```

Kopiering av lister

Kapittel 7.6

Å kopiere lister

- Hvis man skriver **list1 = list2**, …
  - refererer begge variable til samme liste
  - Gjør vi deretter **list2[1] = 9**, vil også list1 være endret



- Hvis man ønsker en uavhengig kopi av ei liste, må man gjøre noe annet… nedenfor er tre ulike alternativer

```
list1 = [1,2,3,4]
list2 = list1[:]       # slice som er hele list1, eller…
# list2 = list(list1)
# list2 = [] + list1
```

To-dimensjonale lister

Kapittel 7.8

To-dimensjonale lister

- To-dimensjonale tabeller brukes i mange sammenhenger
- I Python realiseres 2D-tabeller som lister av lister
  - Ytterste liste: hvert element er en hel rad
  - Innerste liste (kolonnene): hvert element er en verdi
- For å prosessere 2D lister trenger man 2 indekser
- Typisk brukes nøstede løkker til å prosessere dem

2D lister : liste av lister
(kunne også brukt liste av tupler)

```
data = ([  ['land', 'hovedstaden', 'arealet'],
           ['Norge', 'Oslo',  385186],
           ['Sverige', 'Stockholm', 449964]])
```

	Kolonne 0	Kolonne 1	Kolonne 2
Rad 0	’land’	’hovedstaden’	’arealet’
Rad 1	’Norge’	’Oslo’	385186
Rad 2	’Sverige’	’Stockholm’	449964

```
print(data[1][0])
fasit = data[2][1]
data[0][0] = ‘Land’
```

```
# Skriver ut Norge
# fasit blir Stockholm
# Endrer ‘land’ til ‘Land’
```

Lage lister av vilkårlig størrelse

- Fremgangsmåte 1:
  - Først lage tom liste, legge til elementer etter hvert
- Fremgangsmåte 2:
  - Masse-opprette elementer med gitt verdi
    - Bruke ”list comprehensions”
    - Eks.: Lage en 2-dimensjonal 10x10 matrise med 0’er:

```
tabell_10x10 = [[0 for col in range(10)]
                for row in range(10)]
```

    - Eks.: Lage en 3-dimensjonal 3x3x3 matrise med 1’ere:

```
tabell_3d = [[[1 for x in range(3)]
              for y in range(3)]
              for z in range(3)]
```

Tupler

Kapittel 7.9

Tupler

- Tuppel: en ikke-muterbar sekvens (kan ikke endres)
  - Likner ellers på lister
  - Når den er opprettet kan innholdet ikke endres
  - Format: **tuple_name = (item1, item2, …)**
  - Tupler støtter operasjoner slik som lister gjør det
    - Elementer kan hentes med indekser
    - Har metoder som index
    - Innebygde funksjoner som len, min, max
    - Har slicing-uttrykk
    - Har operatorene in, + og *

Kodeeksempel: tone_tuppel.py

Tupler (forts.)

- Tupler støtter ikke endringsmetoder:
  - ~~append, remove, insert, reverse, sort~~
  - Ulempe: mindre fleksible
    - Men noen Python-operasjoner krever tupler
  - Fordel:
    - Programmet kjører raskere
    - Tryggere for data som ikke skal endres (f.eks. ukedager, måneder)
- Funksjonene list() og tuple(): konverterer mellom de to

```
tuppel = (1,2,3)
print(tuppel[0])          # skriver 1
liste = list(tuppel)      # gir liste = [1,2,3]
tuppel2 = tuple(liste)    # gir tuppel2 = (1,2,3)
```

Oppsummering

- Dette kapitlet dekket:
  - Lister
    - Repetisjons- og konkateneringsoperatorer
    - Indeksering
    - Teknikker for å prossesere lister (gå igjennom lister)
    - Å slice (plukke ut deler) og kopiere lister
    - Listemetoder og innebygde funksjoner for lister
    - To-dimensjonale lister
  - Tupler
    - Ikke muterbar (kan ikke endres)
    - Forskjeller fra lister, fordeler og ulemper