NTNU
Kunnskap for en bedre verden

TDT4110 Informasjonsteknologi grunnkurs:
Tema: Lister og tupler
Kap 5: Egne moduler + et større eksempel
Kap. 7

Terje Rydland - IDI/NTNU

Litt vi ikke rakk forrige gang

- Mål
  - Forstå hva en modul er og hvordan man kan bruke moduler
  - Et litt større eksempel
- Pensum
  - Starting out with Pyton:
  - Chapter 5.10

Å lagre funksjoner i moduler Kap 5.10

- I større, kompliserte programmer er det viktig å organisere koden godt
- Modularisering: å gruppere relaterte funksjoner i moduler
  - Gjør at programmene er lettere å forstå, teste og vedlikeholde
  - Gjør det lettere å bruke gode om igjen for flere programmer
    - Importer modulen som inneholder funksjonen til hvert program som trenger den

Vi behøver ikke bare bruke eksisterende moduler

Vi kan skrive våre egne!

Å lagre funksjoner i moduler (forts.)

- En Modul er en fil som inneholder Python-kode
  - Inneholder funksjonsdefinisjoner, men inneholder ikke kall til disse funksjonene
    - Programmer vil importere modulene og kalle funksjonene
- Regler for modulnavn:
  - Filnavn skal slutte med .py
  - Kan ikke være det samme som et nøkkeluttrykk i Python
- Importer moduler med import-uttrykket

Visualisering

Program

```
def dill():
    linje1
    .
    .
def dall():
    linje1
    .
    .
linje1
linje2
.
.
```

Bibliotek

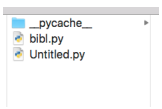
mittBibl.py

```
def dill():
    linje1
    .
    .
def dall():
    linje1
    .
    .
def dull():
    linje1
    .
    .
```


bibl.py - /Users/terjery/Library/Mobile Documents/com~apple~CloudDocs/ITGK 2016/Python/1Forelesninger/Uke 40/ModulEksempel/bibl.py (3.5.2)

```
import math

def sirkelAreal(radius):
    return math.pi*radius**2

def sirkelOmkrets(radius):
    return 2*math.pi*radius

def hypPythagoras(kat1,kat2):
    return math.sqrt(kat1**2 + kat2**2)

def katPythagoras(kat,hyp):
    if hyp < kat:
        print('Dette er umulig. Hypotenusen kan ikke være kortere enn kateten.')
        return 0
    else:
        return math.sqrt(hyp**2 - kat**2)

def rektangel(lengde,bredde):
    omkrets = 2*lengde + 2*bredde
    areal = lengde*bredde
    return omkrets,areal
```

Ln: 1 Col: 0



Untitled.py - /Users/terjery/Library/Mobile Documents/com~apple~CloudDocs/ITGK 2016/Python/1Forelesninger/Uke 40/ModulEksempel/Untitled.py (3.5.2)

```
import bibl

kat = 3
hyp = 7
print('Den manglende siden i trekanten er',bibl.katPythagoras(kat,hyp),'cm.')

l = 34
b = 45
omkrets,areal = bibl.rektangel(l,b)
print('Omkrets:',omkrets,'Areal:',areal)
```

Ln: 1 Col: 0

Større eksempel

- Lag et program som oversetter fra latinske tall til romertall.
- Programmet skal fungere for alle tall fra 0 til 3999

Første iterasjon

- Lag funksjonen `romersiffer`
  - tar inn et tall mellom 0 og 9 og returnerer et romertall som svarer til tallet.
- Romertallene fra 0 til 9 er som følger:

```
0='', 1=I, 2=II, 3=III, 4=IV, 5=V, 6=VI, 7=VII,
 8=VIII, 9=IX
```

- Bruker kan skrive inn et tall og få oversatt det til et romertall som skrives ut til konsollet så lenge at tallet brukeren skriver inn er større eller lik 0.

romertall_1.py

Pseudokode - hovedprogram

Skriv overskrift
Les inn tall fra tastaturet
Så lenge tallet >= 0
    Beregn et romersiffer
    Skriv ut resultatet
    Les inn nytt tall fra tastaturet
Skriv sluttbeskjed

Romertall håndterer kun heltall så vi trenger ikke ta hensyn til desimaltall.

```
print('Programmet konverterer fra latinske tall til romertall')
tall = int(input('Oppgi et tall mellom 0 og 9: '))
while tall >= 0:
    resultat = romersiffer(tall)
    print(tall,'blir',resultat,'som romertall.')
    tall = int(input('Oppgi et tall mellom 0 og 9: '))
print('\nTakk for nå.')
```

Ln: 22 Col: 0

Pseudokode - Beregn romersiffer

Motta tall fra kallende funksjon/hovedprogram
Sjekk tall
    returner riktig romersiffer
    gi feilmelding hvis tallet er større enn 9

*Romertall_1.py - /Users/terjery/Library/Mobile Documents/com~apple~CloudDocs/ITGK 2016/Python/1Forelesninger/Uke 41/Romertall_...

```
def romersiffer(n):
    # returnerer romertall som svarer til n
    if n==0:   return ''
    elif n==1: return 'I'
    elif n==2: return 'II'
    elif n==3: return 'III'
    elif n==4: return 'IV'
    elif n==5: return 'V'
    elif n==6: return 'VI'
    elif n==7: return 'VII'
    elif n==8: return 'VIII'
    elif n==9: return 'IX'
    else: return 'Tallet er for stort'
```


romertall_1.py

*Romertall_1.py - /Users/terjery/Library/Mobile Documents/com~apple~CloudDocs/ITGK 2016/Python/1Forelesninger/...

```
def romersiffer(n):
    # returnerer romertall som svarer til n
    if n==0:   return ''
    elif n==1: return 'I'
    elif n==2: return 'II'
    elif n==3: return 'III'
    elif n==4: return 'IV'
    elif n==5: return 'V'
    elif n==6: return 'VI'
    elif n==7: return 'VII'
    elif n==8: return 'VIII'
    elif n==9: return 'IX'
    else: return 'Tallet er for stort'

print('Programmet konverterer fra latinske tall til romertall')
tall = int(input('Oppgi et tall mellom 0 og 9: '))
while tall >= 0:
    resultat = romersiffer(tall)
    print(tall,'blir',resultat,'som romertall.')
    tall = int(input('Oppgi et tall mellom 0 og 9: '))
print('\nTakk for nå.')
```

Ln: 22 Col: 0

Funksjoner som bruker funksjoner – et større eksempel

- Struktur på tallene som også viser seg i 10 og 100-posisjonen
  - 1 2 3 4 5 6 7 8 9: `I II III IV V VI VII VIII IX X`
  - 10 20 30 40 50 60 70 80 90 : `X XX XXX XL L LX LXX LXXX XC C`
  - 100 200 300 400…1000: `C CC CCC CD D DC DCC DCCC DM M`
- Utnytte det til å skrive **et generelt program** for å generere romertall ut fra et latinsk tall
- Vi har et program som kan konvertere et latinsk siffer (0…9) til et romertall
  - Kan vi utvide det til å håndtere 10ere og 100ere?

Funksjoner som bruker funksjoner – et større eksempel (fortsettelse)

- For å angi det siste sifferet i et heltall som et romertall bruker vi symbolene I=1 og V=5
- For å angi det nest siste sifferet i et heltall som et romertall bruker vi symbolene X=10 og L = 50
- For å angi det nest-nest siste sifferet i et heltall som et romer tall bruker vi symbolene, C=100 og D = 500 etc
- Romertall oppfører seg på samme måte for tall større en 10
  - 0 - 10: I, V og X
  - 10 - 100: X, L og C
  - 100 - 1000: C, D og M

Funksjoner som bruker funksjoner – et større eksempel (fortsettelse)

- Vi skal utvide slik at vi kan skrive et romertall avhengig av hvilket siffer vi skal angi.
- Generalisere det programmet vi nettopp har skrevet

Viktig sammenheng

1	2	3	4	5	6	7	8	9	10
I	I+I	I+I+I	I+V	V	V+I	V+I+I	V+I+I+I	I+X	X
10	20	30	40	50	60	70	80	90	100
X	X+X	X+X+X	X+L	L	L+X	L+X+X	L+X+X+X	X+C	C
100	200	300	400	500	600	700	800	900	1000
C	C+C	C+C+C	C+D	D	D+C	D+C+C	D+C+C+C	C+M	M

Pseudokode - Beregn romersiffer

Motta tall og de tre tegnene som skal brukes fra kallende funksjon/hovedprogram
Sjekk tall
  returner riktig romersiffer
  gi feilmelding hvis sifferet er større enn 9

```
def romersiffer(n):
    # returnerer romertall som svarer til n
    if n==0:  return ''
    elif n==1: return 'I'
    elif n==2: return 'II'
    elif n==3: return 'III'
    elif n==4: return 'IV'
    elif n==5: return 'V'
    elif n==6: return 'VI'
    elif n==7: return 'VII'
    elif n==8: return 'VIII'
    elif n==9: return 'IX'
    else: return 'Tallet er for stort'
```

Dette gir riktig svar for 0..9

Dette gir riktig svar for
0..9
10..90
100..900
avhengig av hva vi sender over som entegn, femtegn etc

```
def romersiffer (n,entegn,femtegn,titegn):
    # returnere romertall som svarer til n
    if n==0:  return ''
    elif n==1: return entegn
    elif n==2: return entegn+entegn
    elif n==3: return entegn+entegn+entegn
    elif n==4: return entegn+femtegn
    elif n==5: return femtegn
    elif n==6: return femtegn+entegn
    elif n==7: return femtegn+entegn+entegn
    elif n==8: return femtegn+entegn+entegn+entegn
    elif n==9: return entegn+titegn
    else: return 'ERROR'
```

romertall_2.py

```
def romersiffer (n,entegn,femtegn,titegn):
    # returnere romertall som svarer til n
    if n==0:    return ''
    elif n==1: return entegn
    elif n==2: return entegn+entegn
    elif n==3: return entegn+entegn+entegn
    elif n==4: return entegn+femtegn
    elif n==5: return femtegn
    elif n==6: return femtegn+entegn
    elif n==7: return femtegn+entegn+entegn
    elif n==8: return femtegn+entegn+entegn+entegn
    elif n==9: return entegn+titegn
    else: return 'ERROR'

def sjekkPosisjon(siffer):
    if siffer == 1:
        return 'I','V','X'
    elif siffer == 10:
        return 'X','L','C'
    else:
        return 'C','D','M'

print('Programmet konverterer fra latinske tall til romertall')
tall = int(input('Oppgi et tall mellom 0 og 9: '))
siffer = int(input('1er, 10er eller 100er (skriv 1, 10 eller 100): '))
en,fem,ti = sjekkPosisjon(siffer)
while tall >= 0:
    resultat = romersiffer(tall,en,fem,ti)
    print(tall,'blir',resultat,'som romertall.')
    tall = int(input('Oppgi et tall mellom 0 og 9: '))
    if tall >= 0:
        siffer = int(input('1er, 10er eller 100er (skriv 1, 10 eller 100): '))
        en,fem,ti = sjekkPosisjon(siffer)

print('\nTakk for nå.')
```

romertall_2.py

Funksjoner som bruker funksjoner – et større eksempel (fortsettelse)

- Vi skal utvide slik at vi kan skrive alle romertall fra 1 t.o.m. 3999
  - 3999 er det største standard romertallet som finnes, da de ikke har en bokstav for 5000 (de bruker $\overline{\text{IV}}$ og $\overline{\text{V}}$ for 4000 og 5000)
- Har allerede funksjon som returnerer rett siffer på 1er, 10er og100-plassen
- Problem:
  - Hvordan kan vi dele opp f.eks. 345 til 300 + 40 + 5
  - Da kan vi oversette hvert siffer og sette dem sammen:
    - 300 = CCC
    - 40 = XL
    - 5 = V
    - 345 = CCCXLV

Pseudokode, oppdeling

Problem:
Hvordan plukke ut siffer for siffer

```
sett romertall = ''
Hvis tall > 3999: Avslutt
sett tusen = tall // 1000
beregn sifferkomb for tusenplassen
r = r + tusenplassiffer
tall = tall - tusen * 1000
sett hundre = tall // 100
beregn sifferkomb for hundreplassen
r = r + hundreplassiffer
tall = tall - hundre * 100
tier = tall // 10
beregn sifferkomb for tierplassen
r = r + tierplassiffer
en = tall % 10
beregn siffer for enerplassen
r += enerplass
```

Justering av hovedprogram

```
print('Programmet konverterer fra latinske tall til romertall')
tall = 0
while tall>=0:
    tall = int(input('Tall mellom 0 og 3999 (negativt tall avslutter): '))
    if n < 0: break
    r = romertall(tall)
    print('Romertall for ',tall,' er ',r)

print('\nTakk for nå.')
```

Funksjonen Romersiffer er den samme

```
def romersiffer (n, entegn, femtegn, titegn):
    # returnerer romertall som svarer til n med
    # spesifiserte tegn for 1, 5 og 10
    if n==0: return ''
    elif n==1: return entegn
    elif n==2: return entegn+entegn
    elif n==3: return entegn+entegn+entegn
    elif n==4: return entegn+femtegn
    elif n==5: return femtegn
    elif n==6: return femtegn+entegn
    elif n==7: return femtegn+entegn+entegn
    elif n==8: return femtegn+entegn+entegn+entegn
    elif n==9: return entegn+titegn
    else: return 'ERROR!!!!!'
```


sett romertall = ''
Hvis tall > 3999: Avslutt
sett tusen = tall // 1000
beregn sifferkomb for tusenplassen
r = r + tusenplass
tall = tall - tusen * 1000
sett hundre = tall // 100
beregn sifferkomb for hundreplassen
r = r + hundreplass
tall = tall - hundre * 100
tier = tall // 10
beregn sifferkomb for tierplassen
r = r + tierplass
en = tall % 10
beregn siffer for enerplassen
r += enerplass

```
def fjern(n,pos):
    return n-(n//pos)*pos

def romertall(n):
    # returere romertall fra 0 til 3999
    r = '' # romertallet som streng
    if n>3999: return 'Tallet er for stort.'
    r = r + romersiffer(n//1000,'M','','')
    n = fjern(n,1000) # Fjerner tusenplassen
    r = r + romersiffer(n//100,'C','D','M')
    n = fjern(n,100) # Fjerner hundeplassen
    r = r + romersiffer(n//10,'X','L','C') # Tierplassen
    r = r + romersiffer(n%10,'I','V','X') # Enerplassen
    return r
```


Hele programmet

```
def romersiffer (n, entegn, femtegn, titegn):
    # returnerer romertall som svarer til n med
    # spesifiserte tegn for 1, 5 og 10
    if n==0: return ''
    elif n==1: return entegn
    elif n==2: return entegn+entegn
    elif n==3: return entegn+entegn+entegn
    elif n==4: return entegn+femtegn
    elif n==5: return femtegn
    elif n==6: return femtegn+entegn
    elif n==7: return femtegn+entegn+entegn
    elif n==8: return femtegn+entegn+entegn+entegn
    elif n==9: return entegn+titegn
    else: return 'ERROR!!!!!'

def fjern(n,pos):
    return n-(n//pos)*pos

def romertall(n):
    # returere romertall fra 0 til 3999
    r = '' # romertallet som streng
    if n>3999: return 'Tallet er for stort.'
    r = r + romersiffer(n//1000,'M','','')
    n = fjern(n,1000) # Fjerner tusenplassen
    r = r + romersiffer(n//100,'C','D','M')
    n = fjern(n,100) # Fjerner hundeplassen
    r = r + romersiffer(n//10,'X','L','C') # Tierplassen
    r = r + romersiffer(n%10,'I','V','X') # Enerplassen
    return r

print('Programmet konverterer fra latinske tall til romertall')
tall = 0
while tall>=0:
    tall = int(input('Tall mellom 0 og 3999 (negativt tall avslutter): '))
    if tall < 0: break
    r = romertall(tall)
    print('Romertall for ',tall,' er ',r)

print('\nTakk for nå.')
```

romertall_3.py

Oppsummering

- Denne presentasjonen dekket:
  - Å bruke bibliotekfunksjoner og import-uttrykket
  - Moduler, inkludert
    - Modulene random og math
    - Å gruppere dine egne funksjoner i moduler

Læringsmål og pensum

- Mål
  - Lære om
    - Sekvenser
      - Lister
      - Tupler
    - List Slicing
    - Finne elementer i lister med operatoren in
    - Liste-metoder og nyttige innebygde funksjoner
    - Kopiere og prosessere lister
    - To-dimensjonale lister
- Pensum
  - Starting out with Pyton:
  - Chapter 7 Lists and Tuples

Sekvenser - Kap. 7.1

- Sekvens: et objekt som inneholder flere dataenheter
  - Enhetene lagres i sekvens (etter hverandre)
- Python har ulike sekvenstyper, inkludert **lister** og **tupler**
  - En liste kan endres både verdien av sine elementer og lenge etter at den er opprettet og kalles **muterbar** (”**mutable**”).
  - En **tuppel** kan ikke endre verdien av sine elementer eller lengde etter at den er oppretter og kalles ”**immutable**”

Lister Kap. 7.2

- I Python brukes liste til å holde på all mulig informasjon som ligger etter hverandre som perler på en snor.
- Innholdet i en liste kan være av typen **tall** (både heltall og flyttall), **sannhetsverdier** (True eller False), **tekst**, **eller en annen liste**.
  - Man kan blande typer i den samme listen
- Funksjonen `print()` kan brukes til å skrive ut innholdet i ei liste og `list()` kan brukes til å omgjøre noen objekter til en liste.

- Opprette liste i Python gjøres på formen:

```
variabelnavn = [element1, element2, element3, …]
```

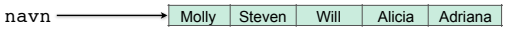
- Eksempel på en liste i Python:

```
liste = ['Ole', 'Dole', 'Doffen']
```


Innledning til lister (forts.)

- En liste av heltall:

- En liste av strenger:



- En liste som inneholder flere ulike datatyper:

kode: various_lists.py

various_lists.py

*various_lists.py - /Users/terjery/Library/Mobile Documents/com~apple~CloudDocs/ITGK 2016/Python/...

```
even_numbers=[2,4,6,8,10]
names=['Molly','Steven','Will','Alicia','Adriana']
info=['Alicia',27,1550.87]
diverse = ['Petter',45,[12,13,14],True]
print(even_numbers)
print(names)
print(info)
print(diverse)
```

Ln: 9 Col: 0

Python 3.5.1 Shell

```
le~CloudDocs/ITGK 2016/Python/Keynote/Uke 41/Python/vario
us_lists.py
[2, 4, 6, 8, 10]
['Molly', 'Steven', 'Will', 'Alicia', 'Adriana']
['Alicia', 27, 1550.87]
['Petter', 45, [12, 13, 14], True]
>>>
```

Ln: 46 Col: 4

Lister med et bestemt antall elementer

- I Python kan man også lage en liste med et bestemt antall elementer ved hjelp av repetisjonsoperatoren `*`.
- Når man skriver en liste etterfulgt av `*` og et heltall, vil man repetere lista like mange ganger som størrelsen på heltallet:

```
liste = [0]*10 # Lager lista [0,0,0,0,0,0,0,0,0,0]
liste = [1,2,3] * 3 # Lager lista [1,2,3,1,2,3,1,2,3]
```

- Ønsker man å lage en liste som er tom, kan man fylle lista med verdien `None`, som er et reservert ord for tomme elementer:

```
liste = [None]*5 # liste med 5 tomme elementer
```

- Vi prøver litt…

kode: repetisjonsoperator.py

repetisjonsoperatorer.py

*repetisjonsoperator.py - /Users/terjery/Library/Mobile Documents/...

```
A = [2,4,6]*3
B = ["Ole","Dole","Doffen"]*2
C= [None]*10
print(A)
print(B)
print(C)
print(A*5)
print(B*2)
print(C*3)
```

Ln: 5 Col: 9

Python 3.5.1 Shell

```
/ITGK 2016/Python/Keynote/Uke 41/Python/repetisjonsoperator.py
[2, 4, 6, 2, 4, 6, 2, 4, 6]
['Ole', 'Dole', 'Doffen', 'Ole', 'Dole', 'Doffen']
[None, None, None, None, None, None, None, None, None, None]
[2, 4, 6, 2, 4, 6, 2, 4, 6, 2, 4, 6, 2, 4, 6, 2, 4, 6, 2, 4, 6, 2, 4, 6, 2, 4, 6, 2, 4, 6, 2, 4, 6, 2, 4, 6, 2, 4, 6]
['Ole', 'Dole', 'Doffen', 'Ole', 'Dole', 'Doffen', 'Ole', 'Dole', 'Doffen', 'Ole', 'Dole', 'Doffen']
[None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None]
>>>
```

Ln: 51 Col: 76

Iterere over ei liste ved hjelp av ei for-løkke

- Man kan gå igjennom en liste element for element (iterere) ved å bruke ei for-løkke på følgende format:

```
for element in liste:
    print(element) # Eller gjør noe annet med element
```

  - liste er navnet på variabelen som inneholder en liste
  - element er en variabel som får verdien av hvert element i lista, element for element

*iterasjon_liste.py - /Users/terjery/Library/Mobile Documents/com...

```
liste=[1,2,3,4,5,6,7,8,9,10]
for i in liste:
    print(i, end = '\t')
print()
```

Ln: 4 Col: 7

kode: iterasjon_liste.py

Oppgave: lister #1

- Skriv Python-koden for å gjøre følgende:
  - Opprett lista [1,3,5,7,9] repetert 5 ganger i variabelen liste
  - Skriv ut hvert element i lista ganget med seg selv

kode: lister_1.py

Oppgave: lister #1

- Skriv Python-koden for å gjøre følgende:
  - Opprett lista [1,3,5,7,9] repetert 5 ganger i variabelen liste
  - Skriv ut hvert element i lista ganget med seg selv

lister_1.py - /Users/terjery/Library/Mobile Documents/com~...

```
liste=[1,3,5,7,9]*5
for i in liste:
    print(i*i)
```

Ln: 1 Col: 0

Indeksering

- Indeks: et tall som spesifiserer et elements posisjon i en liste
  - Gir tilgang til de enkelte elementer i en listen
  - Indeksen til det første elementet er 0, det andre 1, og det n-te elementet er n-1
  - Negative indekser identifiserer posisjoner relativt til listens slutt
    - Indeksen-1 identifiser det siste elementet, -2 det nest siste osv.

Indeks:	0	1	2	3	…	n-3	n-2	n-1
Verdier:	43	95	12	5	…	43	23	54

Indeks:	-n	-n+1	-n+2	-n+3	…	−3	−2	−1
Verdier:	43	95	12	5	…	43	23	54

Hente ut innhold fra et element i en liste

- For å få tak i innholdet for et bestemt element i en liste, skriver man variabelnavnet, firkantparentes og nummeret på elementet, f.eks:

Element nr. 1

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

- Merk at det første elementet er element 0 (ikke 1)

`x = liste[5]` vil gi **feilmelding**

```
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    liste[5]
IndexError: list index out of range
```


Funksjonen len

- En «**exception**» av typen IndexError utløses hvis en ugyldig indeks brukes (kommer tilbake til dette)
- Funksjonen **len**: returnerer lengden til en sekvens, som for eksempel lengden av ei liste

```
size = len(my_list)
```

  - Returnerer **antallet elementer** i listen, slik at det siste elementet er len(list)-1 ettersom indeks starter på 0 og ikke 1
  - Kan brukes til å hindre en IndexError-exception når man itererer gjennom en liste i en løkke

Oppgave: lister #2

- Skriv Python-koden for å gjøre følgende:
  - Opprett lista [1,3,5,7,9] repetert 5 ganger i variabelen liste
  - Skriv ut hvert tredje element i lista ganget med seg selv ved hjelp av for-løkke, range() og len().

kode: lister_2.py

Oppgave: lister #2

- Skriv Python-koden for å gjøre følgende:
  - Opprett lista [1,3,5,7,9] repetert 5 ganger i variabelen liste
  - Skriv ut hvert tredje element i lista ganget med seg selv ved hjelp av for-løkke, range() og len().

lister_2.py - /Users/terjery/Library/Mobile Documents/com~apple~CloudDocs/IT...

```
liste=[1,3,5,7,9]*5
lengde=len(liste)
for x in range(0,lengde,3):
    print(liste[x]*liste[x])
```

Ln: 6 Col: 4

Lister kan endres (muteres)

- Muterbare sekvenser: **elementer i en liste som kan endres**
  - **Lister er muterbare**, dermed kan deres elementer endres
  - `list[1] = new_value` kan brukes til å tilordne en ny verdi til et element i en liste
    - Må bruke en gyldig indeks for å forhindre at en IndexError-exception utløses (dvs. at elementet for gitt indeks eksisterer).
  - Eks på endring av ei liste:

```
liste = [1, 2, 3, 4, 5]
liste[0] = 3 # Endrer element 0 til 3
#Gir liste = [3, 2, 3, 4, 5]
```


Å konkatinere (sette sammen) lister

- Konkatinering: å føye to ting sammen
- Operatoren **+** kan brukes til å konkatinere (sette sammen) to lister

```
A = [1,2,3,4,5]
B = [2,4,6,8,10]
C = A + B # gir [1,2,3,4,5,2,4,6,8,10]
```

- Den utvidede operatoren **+=** kan også brukes til å konkatinere lister (legge til element(er) til lista).

```
A=[1,2,3]
A+=[4] # Gir A=[1,2,3,4] samme som A = A +[4]
```

Oppgave: lister #3

- Skriv Python-koden for å gjøre følgende:
  - Opprett lista [2,4,6,8,10,12,14,16,18,20] i variabelen `liste`
  - Skriv ut lista til skjerm
  - Bytt ut hvert andre element med 3 gangen starter med element med indeks 1 som vil gi lista [2,**3**,6,**6**,10,**9**,14,**12**,18,**15**]
  - Skriv ut lista til skjerm.

kode: lister_3.py

Oppgave: lister #3

lister_3.py - /Users/terjery/Library/Mobile Documents/com~apple~C...

```
liste=[2,4,6,8,10,12,14,16,18,20]
print(liste)
lengde=len(liste)
i=1
for x in range(1,lengde,2):
    liste[x]=i*3
    i+=1
print(liste)
```

Ln: 1 Col: 0

Python 3.5.1 Shell

```
loudDocs/ITGK 2016/Python/Keynote/Uke 41/Python/lister_3.py
[2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
[2, 3, 6, 6, 10, 9, 14, 12, 18, 15]
>>>
```

Ln: 6 Col: 23

Å skive/slice lister Kap 7.3

- Slice: et spenn av enheter som er tatt fra en sekvens
  - Slicing format: **`list[start:slutt:inkrement]`**
  - Spenn er ei liste som inneholder kopier av elementer fra start fram til, men som ikke inkluderer end
    - Hvis start ikke er spesifisert, brukes 0 som startindeks
    - Hvis end ikke spesifiseres brukes len(list) som sluttindeks
  - Slicing-uttrykk kan inneholde stegverdier og negative indekser, som 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

- Man kan endre på innhold på deler av lister ved hjelp av slice på følgene format:

```
liste[start:slutt:inkrement] = […]
```

•Erstatter den delen av lista spesifisert på venstre side av er-lik tegnet med lista spesifisert på høyre side.

```
A=[1,2,3,4,5,6]      # Lager en liste A
A[:2]=[0,0]          # Erstatter to første elementer. Gir A=[0,0,3,4,5,6]
A[-2:] = [9,9]       # Erstatter to siste elementer. Gir A=[0,0,3,4,9,9]
A[0::2] = [5,5,5]    # Erstatter hvert andre element.Gir A=[5,0,5,4,5,9]
A[-3:]=[]            # Erstatter tre siste elementer med 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]
```


Sette inn flere elementer i en liste

- Hvis man ønsker å legge til flere elementer på slutten av ei liste kan man bruke methoden `extend`:

```
A = [1,2,3]
A.extend([4,5,6]) # gir A = [1,2,3,4,5,6]
```

- Hvis man ønsker å legge til flere elementer til en liste på gitt indeks kan man bruke slicing til dette:

```
A=[1,2,3,4,5,6,7,8,9,10]
A[3:3]=[100,101,102]
                # Setter inn tre elementer på indeks 3
A=[1,2,3,100,101,102,4,5,6,7,8,9]
```


Å finne enheter i listen med operatoren *in* Kap 7.4

- Du kan bruke operatoren **in** til å avgjøre om en enhet finnes i en liste
  - Generelt format: `if item in list:`
  - Returnerer **True** hvis enheten er i listen, eller **False** hvis den ikke er der
- Tilsvarende kan du bruke operatoren **not in** for å avgjøre om et element ikke finnes i listen

in_operatoren.py - /Users/terjery/Library/Mobile Documents/com~apple~CloudDocs/ITGK 20...

```
liste=[1,2,3,4,6,8,9,123,43,54,34,65,76,23,54]
svar=int(input("Gjett på et tall i lista: "))
if svar in liste:
    print(svar,"er i lista")
else:
    print(svar,"er ikke i lista")
```

Ln: 1 Col: 0

kode: in_operatoren.py

Listemetoder og nyttige, innebygde funksjoner

- **`append(item):`** brukes til å legge enheter til en liste – item tilføyes til slutt i lista:

```
A=[1,2,3]
A.append(5)   # Gir A=[1,2,3,5]
```

- **`index(item):`** brukes til å finne hvor man finner en enhet i listen
  - Returnerer indeksen til det første elementet i listen som inneholder elementet item
  - Utløser en ValueError-exception hvis item ikke finnes i listen

```
A=[1,2,7,4,3,2]
print(A.index(7)) # Gir resultatet 2
```


Listemetoder, og nyttige, innebygde funksjoner (forts.)

- **`insert(index, item)`**
  - brukes til å smette enheten item inn i listen ved posisjon index i listen
- **`sort()`**
  - brukes til å sortere elementene i listen i stigende rekkefølge
- **`remove(item)`**
  - fjerner den første forekomsten av enheten item i listen
- **`reverse()`**
  - snur rekkefølgen på elementene i listen

Listemetoder og nyttige, innebygde funksjoner (forts.)

- **`del:`** fjerner et element fra en spesifikk indeks i en liste
  - Generelt format

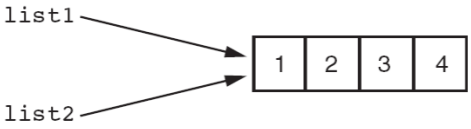
```
del list[i] # Merk ingen paranteser
```

- Funksjonene **min** og **max**: innebygde funksjoner som returnerer enheten som har den laveste eller høyeste verdien i en sekvens
  - Sekvensen sendes som argument
  - Fungerer også på tekststrenger

```
A=[8,3,6,45,12,23,5,65,76,34,2,2]
print(min(A)) # gir 2
print(max(A)) # gir 76
```

Å kopiere lister

- Hvis man skriver **`list1 = list2`**, refererer dette til samme liste.
  - Det vil si at man ikke kopierer og lager en ny liste, men at begge variablene peker til akkurat samme liste.



Å kopiere lister (2)

- For å ta en kopi av en liste så må du kopiere hvert element i listen
  - To metoder gjør dette
    - Å lage en ny, tom liste og bruke en for-løkke til å legge til kopier av hvert element fra den opprinnelige listen til den nye listen

```
liste1 = [1,2,3,4]
liste2 = [] # Lager tom liste
for item in liste1:
        liste2.append(item)
```

    - Å lage en ny, tom liste og konkatinere den gamle listen til den nye

```
liste1 = [1,2,3,4]
liste2 = [] + liste1    # Tom liste +
                                # legger til liste1
```


Å prosessere lister

- Listeelementer kan brukes i beregninger
- For å beregne antallet numeriske verdier i en liste, kan du bruke løkke med en tellende variabel
- For å finne gjennomsnittlig verdi i en liste:
  - Beregne summen av verdier i listen
  - Dividere summen med **`len(list)`**
- Lister kan brukes som argumenter til funksjoner
- En funksjon kan returnere en referanse til ei liste

kode: snitt.py

Å prosessere lister

snitt.py - /Users/terjery/Library/Mobile Documents/com~apple~CloudDocs/ITGK 2016/Python/Keynote/Uke 41/Python/s...

```
def gjennomsnitt(liste):
    summen=0
    for item in liste:
        summen+= item
    gjsnitt = summen/len(liste)
    return gjsnitt

liste=[] # Lager tom liste
tall = -1
while(tall!=0):
    tall=float(input("Skriv inn flere tall (avslutt med 0) "))
    liste.append(tall)

liste.remove(0) # Fjerner siste element med 0

snitt = gjennomsnitt(liste)
print("Gjennomsnittet er:",snitt)
```

Ln: 1 Col: 0

To-dimensjonale lister

- To-dimensjonale liste: liste som inneholder andre lister som elementer
  - Også kjent som nøstede lister
  - Vanlig å betrakte to-dimensjonale lister som om de har rekker og kolonner
  - Nyttig til å jobbe med flere datasett
- For å prosessere data i to-dimensjonale lister trenger man å bruke indekser
- Typisk brukes nøstede løkker til å prosessere dem

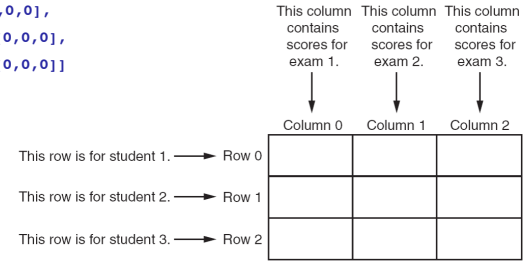
Lage to-dimensjonale lister (liste av lister)

```
students=[      ['Joe','Kim'],
                ['Sam','Sue'],
                ['Kelly','Chris']]
```

	Column 0	Column 1
Row 0	'Joe'	'Kim'
Row 1	'Sam'	'Sue'
Row 2	'Kelly'	'Chris'

Lage to-dimensjonale lister (lister av lister)

```
scores=[[0,0,0],
        [0,0,0],
        [0,0,0]]
```



Hente ut verdier fra to-dimensjonale lister:

	Column 0	Column 1	Column 2
Row 0	scores[0][0]	scores[0][1]	scores[0][2]
Row 1	scores[1][0]	scores[1][1]	scores[1][2]
Row 2	scores[2][0]	scores[2][1]	scores[2][2]

```
print(scores[1][1]) # Skriver ut element (1,1)
x=scores[2][0]    # Setter x lik element (2,0)
scores[0][0] = 9          # Setter element (0,0) lik 9
```


Lage to-dimensjonale tabeller av vilkårlig størrelse

- Man kan også opprette en fler-dimensjonal tabell av en gitt størrelser uten å angi alle elementene:
  - Lage en 2-dimensjonal 10x10 matrise med 0er:

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

  - Lage en 3-dimensjonal 3x3x3 matrise med 1ere:

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

Oppgave: matriser

- Skriv Python-koden for å gjøre følgende:
  - Opprett en 3 x 3 tabell av 0er
  - Fyll tabellen så den blir som følgende:
  - Skriv ut tabellen rekke for rekke

2	4	6
8	10	12
14	16	18

kode: matriser.py

Oppgave: matriser

matriser.py – /Users/terjery/Library/Mobile Documents/com~apple~CloudDocs/ITGK 2016/Python/Key...

```
Tabell=[[0 for x in range(3)] for y in range(3)]

tall=2
for y in range(3):
    for x in range(3):
        Tabell[y][x]=tall
        tall+=2

for rekke in Tabell:
    print(rekke)
```

Ln: 1 Col: 0

Python 3.5.1 Shell

```
ke 41/Python/matriser.py
[2, 4, 6]
[8, 10, 12]
[14, 16, 18]
>>>
```

Ln: 13 Col: 4

Tupler

- Tuppel: en ikke-muterbar sekvens (kan ikke endres)
  - Likner ellers på lister
  - Når den er opprettet kan den 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 *

Tupler (forts.)

- Tupler **støtter ikke** metoder som innebærer endring av sekvensen (naturlig nok):
  - append
  - remove
  - insert
  - reverse
  - sort

Tupler (forts.)

- Fordeler med å bruke tupler fremfor lister:
  - Det går raskere å prossesere dem
  - Tupler er trygge (de kan ikke tukles med)
  - Noen Python-operasjoner krever tupler
- Funksjonen list(): gjør tuppel om til liste

```
tuppel = (1,2,3)
liste = list(tuppel) # gir liste = [1,2,3]
```

- Funksjonen tuple(): gjør liste om til tuppel

```
liste=[4,5,6]
tuppel = tuple(liste) # gir tuppel = (4,5,6)
```


Oppsummering

- Dette kapittelet dekket:
  - Lister, som i
    - Repetisjons- og konkatineringsoperatorer
    - 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, som i
    - Ikke muterbar (kan ikke endres)
    - Forskjeller fra lister, og fordeler fremfor lister