NTNU

Kunnskap for en bedre verden

Kunnskap for en bedre verden

TDT4110 Informasjonsteknologi grunnkurs:

Tema: Mer om strenger

Utgave 3: Kap. 8

Terje Rydland - IDI/NTNU

Læringsmål og pensum

- Mål
  - Lære om
    - Slicing av lister
    - 2-dimensjonale lister
- Pensum
  - Starting out with Python:
  - Chapter 7 Lists

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
  - Konkatineringsoperatoren: +
- 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]
```

Større eksempel

hangman.py - /Users/terjery/Library/Mobile Documents/com~apple~CloudDocs/ITGK 2016...

```
def skrivUtListe(liste):
    for i in liste:
        print(i,end='')
    print('\n')

print('\nHangman. Prøv å gjette ordet.')
tekst = 'Hemmelig'
liste = ['*']*len(tekst)
skrivUtListe(liste)
n = 0
while list(tekst) != liste:
    valg = input('Gjett en bokstav: ')
    if valg in tekst:
        print('Bokstaven finnes i ordet.')
        for i in range(len(liste)):
            if tekst[i] == valg:
                liste[i] = valg
        skrivUtListe(liste)
    else:
        print('Bokstaven finnes ikke i ordet.')
        n += 1
print('Du hadde',n,'feil.')
```

Ln: 6 Col: 9

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]
                  # tilsvarer A += [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 - *bruk dot-notasjon*

- **`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) # liste(5,3)`**
  - brukes til å smette enheten item inn i listen ved posisjon index i listen
- **`sort()  # liste.sort()`**
  - brukes til å sortere elementene i listen i stigende rekkefølge
- **`remove(item)  # liste.remove(3)`**
  - fjerner den første forekomsten av enheten item i listen
- **`reverse() # liste.reverse()`**
  - snur rekkefølgen på elementene i listen

Nyttige, innebygde funksjoner

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

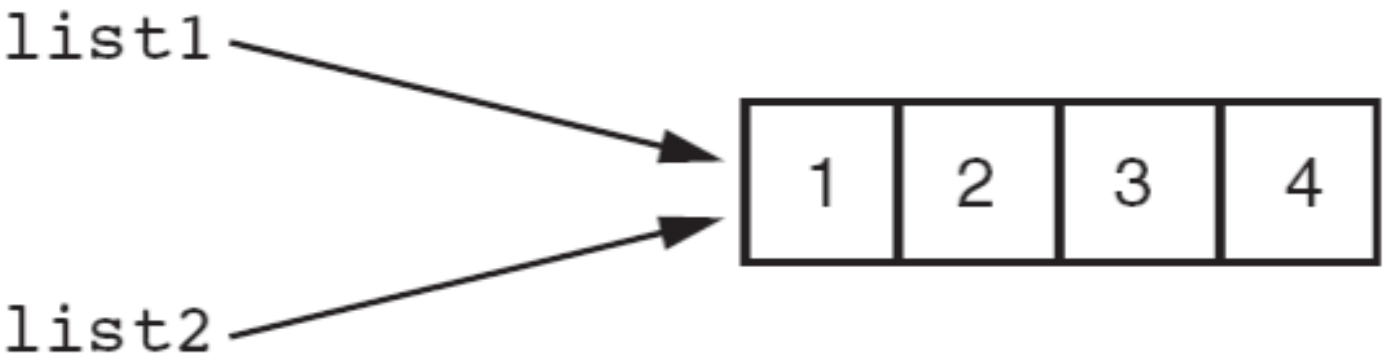
```
del list[i] # Merk ingen paranteser
```

- 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

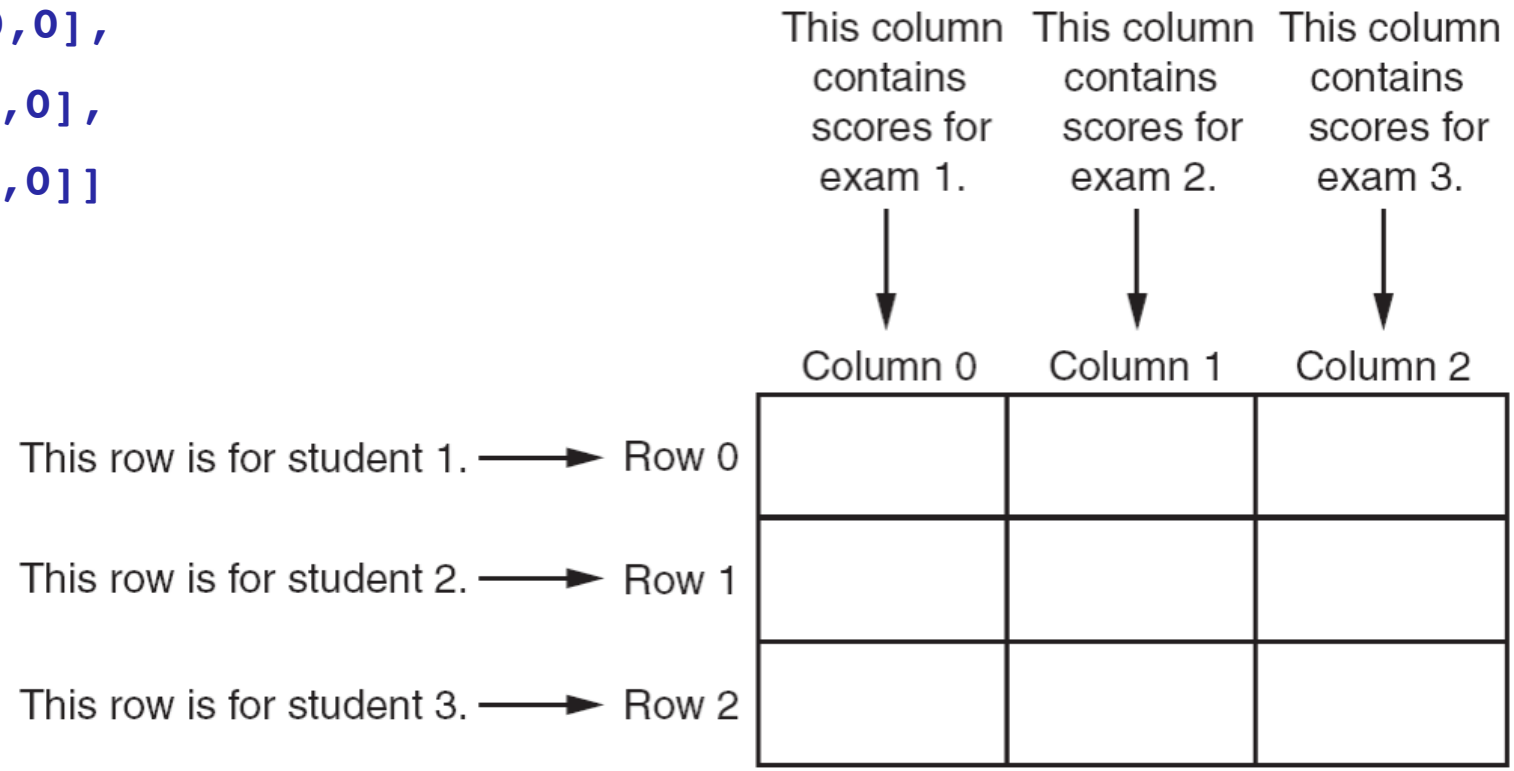
Lage to-dimensjonale lister (liste av lister)

```
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

Læringsmål og pensum

- Mål
  - Lære om
    - Grunnleggende operasjoner på strenger
    - Å skive/slice strenger
    - Teste, søke i og manipulere strenger
- Pensum
  - Starting out with Python:
  - Chapter 8 More About Strings

Tekststrenger (string)

- I Python kan en tekststreng sees på som en liste av tegn og bokstaver med fast lengde.
- En viktig forskjell på liste og tekststreng er at tekststreng ikke kan endre type eller deler av innhold.
- På samme måte som for lister, kan man få ut deler av strengen ved å bruke indeks og slice:

```
tekst = 'Dette er en test'
tekst[0] # Gir 'D'
tekst[14] # Gir 's'
```

Grunnleggende strengoperasjoner

- Mange slags programmer utfører operasjoner på strenger
- I Python finnes mange verktøy for å undersøke og manipulere strenger
  - Strenger er sekvenser, så mange av verktøyene som fungerer med andre sekvenser fungerer med strenger

Tilgang til enkelttegn i en streng

- For å få tilgang til de enkelte tegn i en streng
  - Bruk en for-løkke
    - Format: `for character in string:`
    - Nyttig når man skal iterere over en hel streng, eksempelvis for å telle antall forekomster av et visst tegn i den
  - Bruk indeksering
    - Hvert tegn har en indeks som spesifiserer dens plassering i strengen, og begynner med 0
    - Format

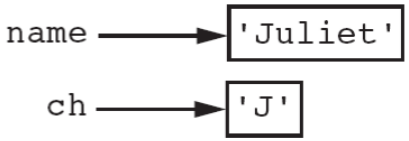
```
character = my_string[i]
```

kode: sjekk_streng.py

Iterasjon over strengen "Juliet"

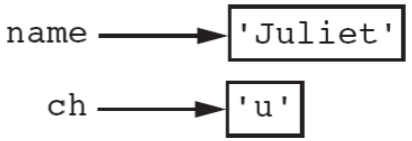
1st Iteration

```
for ch in name:
    print(ch)
```



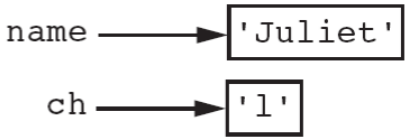
2nd Iteration

```
for ch in name:
    print(ch)
```



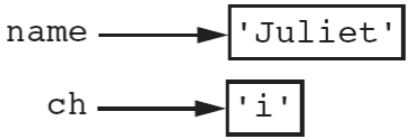
3rd Iteration

```
for ch in name:
    print(ch)
```



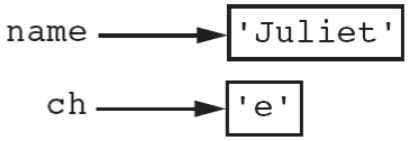
4th Iteration

```
for ch in name:
    print(ch)
```



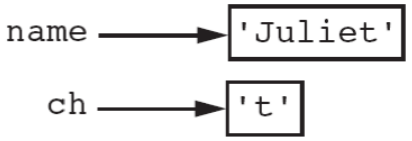
5th Iteration

```
for ch in name:
    print(ch)
```



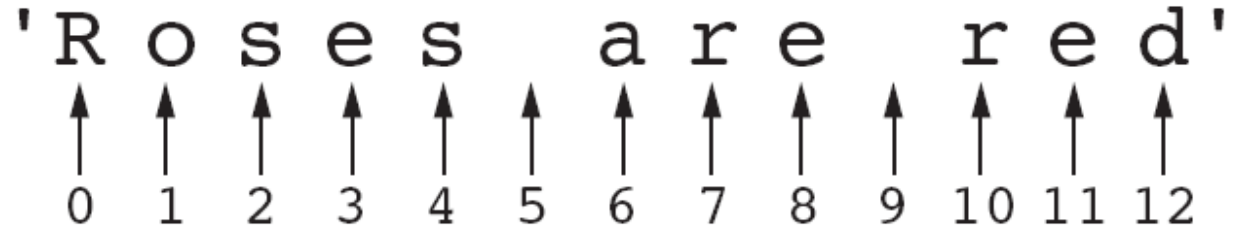
6th Iteration

```
for ch in name:
    print(ch)
```

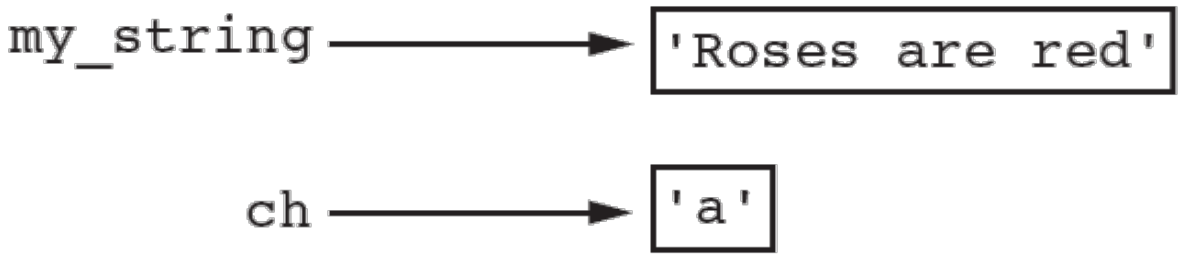


Tilgang til enkelttegn i en streng (forts.)

- `my_string = 'Roses are red'`



- `ch = my_string[6]`



Tilgang til enkelttegn i en streng (forts.)

- IndexError-exception utløses hvis:
  - Du prøver å bruke en indeks som er utenfor strengens rekkevidde
    - Dette skjer nok hvis du forsøker å iterere videre når strengen slutter
- Funksjonen `len(string)` kan brukes til å finne strengens lengde på samme måte som for lister
  - Snedig for å hindre at man itererer forbi strengens slutt

Konkatinering av strenger

- Konkatinering: å tilføye en streng ved slutten av en annen
  - Bruk operatoren + til å lage en streng som er en kombinasjon av dens operander
  - Den utvidede tilordningsoperatoren += kan også brukes til å sette sammen strenger
    - Operanden på venstre side av operatoren += må være en eksisterende variabel; i motsatt tilfelle utløses en exception

Tekststrenger og konkatenering (sammensetn)

- Men man kan sette sammen flere tekststrenger ved å bruke + i mellom tekststrengene:

```
tekst = 'Dette '+ 'er ' + 'en ' + 'test'
# Gir 'Dette er en test'
```

- Kan også sette sammen tekststrenger fra variabler:

```
nytekst = 'Her kommer: '+ tekst
# Gir 'Her kommer: Dette er en test'
```

- Hvis variabler med tall skal settes inn en streng, bruk str(variabel) som gjør om variabel til tekststreng:

```
tekst = 'Tallet er: '+ str(tall)
```

Eksempel

- input-funksjonen kan f.eks. bare ha en parameter.
  - Noen ganger vil vi ha flere i input-funksjonen
  - Bruk konkatenering

09eks.py - /Users/terjery/Library/Mobile Documents/com~apple~CloudDocs/ITGK 2016/Python/PDF/Ekse...

```
summ = 0
for i in range(10):
    tekst = 'Oppgi tall nr ' + str(i+1) + ': '
    tall = int(input(tekst))
    summ += tall

print(summ)
```

Ln: 8 Col: 0

Strenger er ikke-muterbare

- Når de først er laget, kan de ikke endres
  - Konkatinering endrer ikke faktisk noen av strengene, men lager tvert i mot en ny streng, og tilordner denne til en av de allerede eksisterende variablene
  - Man kan **ikke** bruke et uttrykk som dette:

```
string[index] = new_character

tekst[3] = "Klare" # Gir feilmelding!!!
```

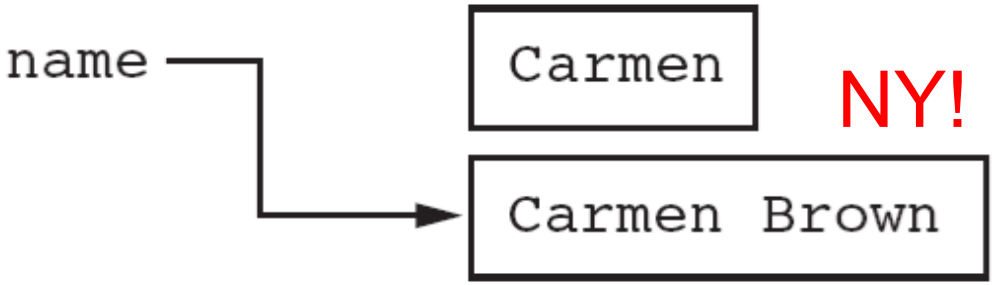
Strenger er ikke-muterbare (forts.)

- Tekststrengen `'Carmen'` blir tilordnet `name`:

```
name = 'Carmen'
```

- Tekststrengen `'Carmen Brown'` blir tilordnet `name`.
  Merk! Et nytt tekst objekt blir laget!!!

```
name = name + ' Brown'
```



Slicing av strenger (som for lister)

- Slice: spennvidde av enheter tatt fra en strengsekvens, kjent som en *substring*
  - Format: **`string[start : end : steg]`**
    - Uttrykket vil returnere en streng som har en kopi av tegnene fra start til, men ikke medregnet end
    - Hvis start er uspesifisert, antas indeks 0
    - Hvis end er uspesifisert, antas indeks **`len(string)`**

```
tekst = 'Dette er en test'

tekst[0:3] # Gir 'Det'. Samme som tekst[:3]

tekst[12:16] # Gir 'test'. Samme som tekst[12:]

tekst[::2] # Gir 'Dtee nts'. (annenhvert tegn)
```

Testing, søking og manipulering av strenger

- Du kan bruke operatoren in til å avgjøre om en streng inneholdes av en annen streng (samme som lister)
  - Generelt format: **`streng1 in streng2`**
    - streng1 og streng2 kan være string literals eller variable som refererer til strenger
  - Tilsvarende kan du bruke operatoren not in til å avgjøre om en streng ikke er inneholdt av en annen streng

Strengmetoder

- Strenger i Python har mange metoder, inndelt på typer av operasjoner
  - Generelt format:
    - `mystring.method(arguments)`
  - Noen metoder tester en streng for spesifikke karaktertrekk
    - Disse metodene er såkalte boolske metoder som returnerer enten `True` hvis betingelsen er sann ellers `False`.

Noen strengteste-metoder:

Method	Description
`isalnum()`	Returns true if the string contains only alphabetic letters or digits and is at least one character in length. Returns false otherwise.
`isalpha()`	Returns true if the string contains only alphabetic letters, and is at least one character in length. Returns false otherwise.
`isdigit()`	Returns true if the string contains only numeric digits and is at least one character in length. Returns false otherwise.
`islower()`	Returns true if all of the alphabetic letters in the string are lowercase, and the string contains at least one alphabetic letter. Returns false otherwise.
`isspace()`	Returns true if the string contains only whitespace characters, and is at least one character in length. Returns false otherwise. (Whitespace characters are spaces, newlines (`\n`), and tabs (`\t`).
`isupper()`	Returns true if all of the alphabetic letters in the string are uppercase, and the string contains at least one alphabetic letter. Returns false otherwise.

Strengmetoder (forts.)

- Noen metoder returnerer en kopi av en streng som det er gjort forandringer på
  - De ”simulerer” at strenger er muterbare
- Sammenlikning av strenger foregår case-sensitivt
  - Store bokstaver skilles fra små bokstaver
  - lower og upper kan brukes til å gjøre sammenlikning av strenger uten å ta hensyn til case

Strengmodifiserings-metoder:

Method	Description
`lower()`	Returns a copy of the string with all alphabetic letters converted to lowercase. Any character that is already lowercase, or is not an alphabetic letter, is unchanged.
`lstrip()`	Returns a copy of the string with all leading whitespace characters removed. Leading whitespace characters are spaces, newlines (`\n`), and tabs (`\t`) that appear at the beginning of the string.
`lstrip(char)`	The *`char`* argument is a string containing a character. Returns a copy of the string with all instances of *`char`* that appear at the beginning of the string removed.
`rstrip()`	Returns a copy of the string with all trailing whitespace characters removed. Trailing whitespace characters are spaces, newlines (`\n`), and tabs (`\t`) that appear at the end of the string.
`rstrip(char)`	The *`char`* argument is a string containing a character. The method returns a copy of the string with all instances of *`char`* that appear at the end of the string removed.
`strip()`	Returns a copy of the string with all leading and trailing whitespace characters removed.
`strip(char)`	Returns a copy of the string with all instances of *`char`* that appear at the beginning and the end of the string removed.
`upper()`	Returns a copy of the string with all alphabetic letters converted to uppercase. Any character that is already uppercase, or is not an alphabetic letter, is unchanged.

kode: strenger2.py

Sammenlikningsmetoder

- Programmer trenger ofte å søke etter substrenger
- Flere metoder finnes for å gjøre dette:
  - `endswith(substring)`: sjekker om en streng slutter med substring
    - Returnerer `True` eller `False`
  - `startswith(substring):` sjekker om en streng begynner med substring
    - Returnerer `True` eller `False`

Søk- og erstatt-metoder

- Flere metoder finnes (forts.)
- **`find(substring)`**: leter etter substring i strengen
  - Returnerer den laveste indeksen av substrengen, eller -1 om substrengen ikke finnes i strengen
- **`replace(substring,new_string)`**:
  - Returnerer en kopi av strengen hvor alle forekomster av substring er erstattet med new_string

Over sikt over søk- og erstattmetoder

Method	Description
`endswith(substring)`	The *substring* argument is a string. The method returns true if the string ends with *substring*.
`find(substring)`	The *substring* argument is a string. The method returns the lowest index in the string where *substring* is found. If *substring* is not found, the method returns −1.
`replace(old, new)`	The *old* and *new* arguments are both strings. The method returns a copy of the string with all instances of *old* replaced by *new*.
`startswith(substring)`	The *substring* argument is a string. The method returns true if the string starts with *substring*.

kode: strenger2.py

Over sikt over søk- og erstattmetoder

strenger2.py - /Users/terjery/Library/Mobile Documents/com~apple~CloudDocs/ITGK 2016/Python/Keynote/Uke 42/Python/str...

```
print("Avslutt programmet med å skrive 'stopp'")
svar = ""
while(svar!="stopp"):
    svar = input("\n\nSkriv inn tekst: ")
    lengde = len(svar)
    print("Første halvdel er:",svar[:lengde//2])
    print("Annenhver bokstav:",svar[::2])
    print("I store bokstaver:",svar.upper())
    print("Kun bokstaver?",svar.isalpha())
    print("Kun tall?",svar.isdigit())
    print("Erstatte 'er' med 'var':",svar.replace("er","var"))

print("\nEndelig ferdig med dette tullet, ble nesten gal!")
```

Ln: 1 Col: 0

Å splitte opp en streng

- Metoden **split**: returnerer en liste som inneholder ordene i strengen
  - Bruker space (mellomrom) som skilletegn som standard
  - Kan bruke et annet skilletegn gjennom å sende det som argument til metoden split
  - Eks:

```
tekst = 'Dette er en test'
print(tekst.split())
```

  - Gir:

```
['Dette', 'er', 'en', 'test']
```

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