

**TDT4109 Informasjonsteknologi grunnkurs:**

**Tema: Sekvenser**

Lister og tupler, del 2

Mer om tekststrenger

Børge Haugset

# To-dimensjonale lister

Kapittel 7.8

## 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]]
```

This row is for student 1. → Row 0

This row is for student 2. → Row 1

This row is for student 3. → Row 2

This column contains scores for exam 1.



Column 0



Column 1



Column 2


## 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 størrelse 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

# Tupler

## Kapittel 7.9

# Tupler

Tuppler: 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
- Tilordning til indeksert element, f.eks. `tuppel[1] = 5`, vil gi programfeil.

# Tupler (forts.)

Fordeler med å bruke tupler fremfor lister:

- Det går raskere å prosessere 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 sekvens om til tuppel

```
tuple(range(1,6)) # gir (1,2,3,4,5)
liste=[6,7,8]
tuppel = tuple(liste) # gir tuppel = (6,7,8)
```

# Generering av store lister eller tupler

Hvordan lage en liste av svært mange tall som teller oppover?

Liste med tall fra 1 til 1000: [1,2,3,4,5...998,999,1000]:

```
liste = [i+1 for i in range(1001)]
```

Eller: `liste = list(range(1, 1001))`

Liste med 3-gangen opp til 1002: [3,6,9,12...999,1002]:

```
liste = [i+3 for i in range(0, 1003, 3)]
```

Liste med kvadratet til 2-gangen opp til 198\*198:

```
liste = [i*i for i in range(2, 200, 2)]
```

NB! Hvis vi ikke vil endre noe på listeverdiene, så er det vanligvis bedre å beholde dem som "generatorer", f.eks. så er det mer effektivt (raskere, mindre minne) med

heller `for x in range(1000)` enn `for x in list(range(1000))`

*tid.py*



# Oppsummering av lister

- Lister:
  - Repetisjons- og kontateneringsoperatorer
  - Indeksering
  - Teknikker for å prosessere 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, og fordeler fremfor lister



# Tekststrenger

## 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 (3rd edition)

# Tekststrenger (string)

## . Ikke-muterbar!

- Prosessering av tekst er en viktig del av programmering.
- I Python er en *tekststreng* en egen datatype.
- Det er en type *sekvens*, så vi kan mange av tingene som lister/tupler:

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



# Grunnleggende strengoperasjoner

## Kapittel 8.1

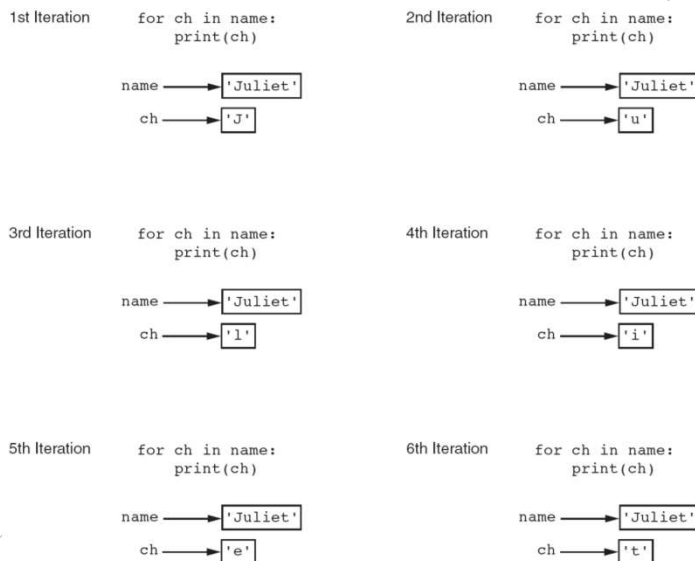
# Grunnleggende strengoperasjoner

- 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]`
- Lag funksjon `telle_tegn` som sjekker antall ganger et tegn befinner seg i en tekst.

# Iterasjon over strengen "Juliet"



## Tilgang til enkelttegn i en streng (forts.)

```
my_string = 'Roses are red'
```

```
'R o s e s   a r e   r e d'
```

↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑

0 1 2 3 4 5 6 7 8 9 10 11 12

```
ch = my_string[6]
```

```
my_string → 'Roses are red'
```

```
ch → 'a'
```

## 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
  - Passer perfekt inn i `range(len(string))`

# Konkatenering (sammenkjeding) av strenger

# Konkatenering av strenger

- konkatenering: 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 konkatenerer strenger
    - Operanden på venstre side av operatoren += må være en eksisterende variabel; hvis ikke 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)
```

## Strenger er ikke-muterbare

- Når de først er laget, kan de ikke endres
  - konkatenering 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!!!
```

    - Dette vil gi en feilmelding

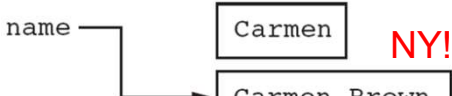
## Strenger er ikke-muterbare (forts.)

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

```
name = 'Carmen'
```


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

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



## Oppgave: finn tegn i tekst



- Skriv Python-koden til funksjonen `skriv_tegn_indeks`, som tar inn en parameterne `tekst` og `tegn`, og skriver ut alle indeksene der `tegn` befinner seg i strengen `tekst`.

kode: `skriv_tegn_indeks.py`

# Slicing (skiving) av strenger

## Kapittel 8.2

# 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

## Kapittel 8.3

# 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
<code>isalnum()</code>	Returns true if the string contains only alphabetic letters or digits and is at least one character in length. Returns false otherwise.
<code>isalpha()</code>	Returns true if the string contains only alphabetic letters, and is at least one character in length. Returns false otherwise.
<code>isdigit()</code>	Returns true if the string contains only numeric digits and is at least one character in length. Returns false otherwise.
<code>islower()</code>	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.
<code>isspace()</code>	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 ( <code>\n</code> ), and tabs ( <code>\t</code> ).
<code>isupper()</code>	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
<code>lower()</code>	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.
<code>lstrip()</code>	Returns a copy of the string with all leading whitespace characters removed. Leading whitespace characters are spaces, newlines ( <code>\n</code> ), and tabs ( <code>\t</code> ) that appear at the beginning of the string.
<code>lstrip(char)</code>	The <i>char</i> argument is a string containing a character. Returns a copy of the string with all instances of <i>char</i> that appear at the beginning of the string removed.
<code>rstrip()</code>	Returns a copy of the string with all trailing whitespace characters removed. Trailing whitespace characters are spaces, newlines ( <code>\n</code> ), and tabs ( <code>\t</code> ) that appear at the end of the string.
<code>rstrip(char)</code>	The <i>char</i> argument is a string containing a character. The method returns a copy of the string with all instances of <i>char</i> that appear at the end of the string removed.
<code>strip()</code>	Returns a copy of the string with all leading and trailing whitespace characters removed.
<code>strip(char)</code>	Returns a copy of the string with all instances of <i>char</i> that appear at the beginning and the end of the string removed.
<code>upper()</code>	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.

# 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

# Oversikt over søk- og erstattmetoder

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

# 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']
```

## Oppgave: overskrift



- Lag funksjonen `overskrift`, som tar inn en parameteren `tekst` og returnerer samme tekststreng der alle ordene starter med stor forbokstav.
- Benytt: `split()`, `upper()`, for-løkke

kode: `overskrift.py`

## Oppgave: finn alle forekomster av delstrenger



- `find` har to parametre til: start og end
- (default verdier: 0 og lengden til strengen)
- `Tekststreng(delstreng, start, slutt)` vil bare søke fra start til slutt-1.
- Lag en funksjon som finner alle startindeks til alle forekomster av delstreng.

kode: `find_all.py`

# Oppsummering av mer om strenger

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