Repetisjon/oppgaver i BED-1304 Python-Lab

Markus J. Aase

Forelesning 4 - Lister, Tuple, Dictionary, NumPy

Informasjon

Dette dokumentet er ment for å repetere og teste forståelsen av kjernepensum i **BED-1304 Python-Lab**. Denne gangen handler det om **Forelesning 4**, som dekker følgende temaer:

- Lister
- Tuple
- Dictionary
- NumPy

I tillegg til definisjoner og eksempler, vil du få oppgaver som kombinerer temaene fra tidligere forelesninger (variabler, uttrykk, funksjoner) med dagens tema. I dette dokumentet er temaene grundig gjennomgått.

Repetisjon og teori

Lister

En liste er en samling av elementer som kan endres (mutable). Lister defineres med hakeparenteser `[]`.

```
frukt = ["eple", "banan", "appelsin"]
print(frukt[0])  # Output: eple

frukt.append("granateple")
print(frukt)  # Output: ["eple", "banan", "appelsin", "granateple"]
```

Viktige metoder:

Viktige metoder for lister i Python

- `append(x)` – Legger til et element på slutten.

```
    liste = [1, 2, 3]
    liste.append(4)
    print(liste)  # [1, 2, 3, 4]

```

- `extend(iterable)` – Legger til alle elementer fra en *iterable* (f.eks. en annen liste).

```
    liste = [1, 2, 3]
    liste.extend([4, 5])
    print(liste)  # [1, 2, 3, 4, 5]

```

- `insert(i, x)` – Setter inn et element (f.eks. `x`) på en gitt posisjon (indeks `i`).

```
    liste = [1, 3, 4]
    liste.insert(1, 2)
    print(liste)  # [1, 2, 3, 4]

```

- `remove(x)` – Fjerner første element som er lik `x`.

```
    liste = [1, 2, 2, 3]
    liste.remove(2)
    print(liste)  # [1, 2, 3]

```

- `pop(i)` – Fjerner og returnerer element på gitt indeks (eller siste hvis ingen indeks oppgis).

```
    liste = [1, 2, 3]
    liste.pop()      # Fjerner 3
    liste.pop(0)     # Fjerner 1
    print(liste)     # [2]

```

- `clear()` – Tømmer hele listen.

```
    liste = [1, 2, 3]
    liste.clear()
    print(liste)  # []

```

- `index(x)` – Returnerer posisjonen til første forekomst av `x`.

```
    liste = [10, 20, 30]
    print(liste.index(20))  # 1

```

- `count(x)` – Teller antall ganger `x` finnes i listen.

```
    liste = [1, 2, 2, 3]
    print(liste.count(2))  # 2

```

- `sort()` – Sorterer listen, i stigende rekkefølge.

```
    liste = [3, 1, 2]
    liste.sort()
    print(liste)  # [1, 2, 3]

```

Hva om lista inneholder tre ord, og ikke heltall? Prøv da vel!

- `reverse()` – Reverserer rekkefølgen.

```
    liste = [1, 2, 5, 3]
    liste.reverse()
    print(liste)  # [3, 5, 2, 1]

```

Hva om lista inneholder tre strings (`str`)? Prøv da vel!

- `copy()` – Returnerer en kopi av listen.

```
    liste = [1, 2, 3]
    kopi = liste.copy()
    print(kopi)  # [1, 2, 3]

```

Slicing av lister i Python

I Python kan vi hente ut deler av en liste ved hjelp av **slicing**. Syntaksen er:

`liste[start:slutt:steg]`

- `start` = indeksen der slicing begynner (inkludert).
- `slutt` = indeksen der slicing slutter (ekskludert).
- `steg` = hvor mye vi hopper mellom elementene (standard er 1).

La oss se på et eksempel:

```
tall = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

print(tall[2:6])      # [2, 3, 4, 5]
print(tall[:4])       # [0, 1, 2, 3]
print(tall[5:])       # [5, 6, 7, 8, 9]
print(tall[::2])      # [0, 2, 4, 6, 8]
print(tall[::-1])     # [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```

Forklaring av eksemplene

- `tall[2:6]` gir elementene fra indeks 2 opp til (men ikke inkludert) 6.
- `tall[:4]` gir de første fire elementene, siden start er tom tolkes den som 0.
- `tall[5:]` gir alle elementer fra indeks 5 og ut listen.
- `tall[::2]` hopper annenhver verdi (her 0, 2, 4, 6, 8).
- `tall[::-1]` reverserer listen.

Tips

- Negativ indeks betyr telling fra slutten (`-1` er siste element).
- `liste[:]` er en enkel måte å lage en kopi av en liste på.

Tuple

En **tuple** ligner på en liste, men den er **immutable**, dvs. at den ikke kan endres etter at den er opprettet. Dette betyr at man ikke kan legge til, fjerne eller endre elementer i en tuple. Man oppretter en tuple med parenteser `()`.

```
koordinat = (10, 20)
print(koordinat[0])   # Output: 10
```

Hvorfor bruke tuple i stedet for liste?

- Når dataene ikke skal endres (f.eks. koordinater, konstanter eller faste innstillinger).
- Tupler er **raskere** å opprette og bruke enn lister, fordi Python kan optimalisere lagringen når innholdet er uforanderlig.
- Tupler kan brukes som **nøkler i dictionaries**, noe lister ikke kan, siden nøkler må være immutable.
- Hvis du skal dele kode med noen, gjør du det tydelig til andre kodere om at dataene er «konstante».

Eksempler på oppretting

```
# Ulike vis man opprette tuples:
t1 = (1, 2, 3)              # Vanlig tuple
t2 = ("a", "b", "c")        # Med strenger
t3 = (1, "a", True)         # Kan blande ulike datatyper
t4 = (42,)                  # En tuple med kun ett element, legg merke til
    kommaet!
t5 = tuple([1, 2, 3, 4])    # Lage tuple fra en liste
```

Tilgang og slicing

Akkurat som med lister kan man bruke indeksering og slicing for å hente ut elementer.

```
t = (10, 20, 30, 40)

print(t[1])       # 20 (indeks 1 - siden Python er null-indeksert)
print(t[-1])      # 40 (siste element)
print(t[1:3])     # (20, 30) - slicing gir og en tuple
```

Pakk ut tuples

En praktisk funksjon er **unpacking**, der man kan tilordne elementene i en tuple direkte til variabler:

```
person = ("Anna", 25, "Oslo")

navn, alder, by = person
print(navn)   # Anna
print(alder)  # 25
print(by)     # Oslo
```

NB: Dette kan du gjøre med lister også! Prøv selv:)

Oppsummering

Bruk lister når data skal endres (eller ønsker å ha muligheten), og tupler når data skal være uforanderlige. Dette gir mer effektiv kode og gjør hensikten med dataene tydeligere.

Dictionary

Et **dictionary** ("ordbokeller "oppslagpå norsk) lagrer data i `key:value`-par (`nøkkel:verdi`-par). Dette gjør det enkelt og effektivt å slå opp verdier basert på en unik nøkkel. Nøkler må være **immutable** (for eksempel `str`, `int`, `tuple`), mens verdiene kan være av hvilken som helst datatype.

```
# Eksempel dictionary
alder = {"Ola": 25, "Kari": 30, "Per": 22}

print(alder["Ola"])  # Output: 25

# Oppdatering av verdi
alder["Kari"] = 31

# Legge til nytt element
alder["Nina"] = 28

print(alder)
# {'Ola': 25, 'Kari': 31, 'Per': 22, 'Nina': 28}
```

NB: Python er det vi kaller *case-sensitive*, som betyr at den er følsomsom om vi skriver en stor bokstav eller liten bokstav. I eksempelet over, skriver du

`alder["ola]`

vil du få en feilmelding fordi nøkkelen er `Ola`, med stor O.

Bruksområder

- Når du trenger å koble nøkkelord (f.eks. navn, ID, produktkode) til bestemte verdier.
- Når rekkefølgen ikke er like viktig som oppslagshastigheten.
- Når du ønsker rask tilgang til data via en unik nøkkel.

Nyttige metoder

- `keys()` – returnerer alle nøkler.
- `values()` – returnerer alle verdier.
- `items()` – returnerer par av `(key, value)`.
- `get(key, value)` – henter verdien til en nøkkel (`key`), returnerer `value` hvis nøkkelen ikke finnes. **OBS:** `value` er valgfri, lar du den stå tom, om nøkkelen ikke finnes, får du `None`.
- `pop(key)` – fjerner og returnerer verdien til en nøkkel.

```
student = {"navn": "Lise", "alder": 21, "studie": "Okonomi"}

print(student.keys())      # dict_keys(['navn', 'alder', 'studie'])
print(student.values())    # dict_values(['Lise', 21, 'Okonomi'])
print(student.items())     # dict_items([('navn', 'Lise'), ('alder', 21), ('
    studie', 'Okonomi')])

# Bruke get() - da det hindre feil
print(student.get("klasse", "Ukjent"))  # Output: Ukjent

# Pop
student.pop("alder")
print(student)  # {'navn': 'Lise', 'studie': 'Okonomi'}
```

Eksempel: telefonbok

```
telefonbok = {
    "Anna": "12345678",
    "Ola": "87654321",
    "Clara": "11223344"
}

print(telefonbok["Anna"])  # 12345678

# Legge til ny kontakt
telefonbok["David"] = "99887766"

# Endre nummer
telefonbok["Clara"] = "44332211"
```

Oppsummering

Dictionaries er svært nyttige for å organisere og lagre data i par. De gir rask tilgang til informasjon og er et av de mest brukte datastrukturene i Python.

NumPy

NumPy er et kraftig Python-bibliotek for numeriske beregninger. Det er spesielt viktig for effektiv håndtering av store datasett, vektorer og matriser. NumPy er mye raskere enn vanlige Python-lister fordi det er bygget oppå optimalisert C-kode.

Import og alias

NumPy importeres vanligvis slik:

```
import numpy as np
```

Her gir vi biblioteket et **alias** (`np`). Dette er en forkortelse som gjør koden mer lesbar og konsis. I stedet for å skrive `numpy.array()`, kan vi nå skrive `np.array()`.

OBS: Dette funker hvis, og bare hvis, man har lastet ned numpy først (ved hjelp av `pip`).

En-dimensjonale og to-dimensjonale arrays i NumPy

NumPy-arrays er grunnlagetfor effektiv numerisk beregning i Python. En **en-dimensjonal array** ligner på en liste, mens en **to-dimensjonal array** ligner på en matrise eller tabell.

En-dimensjonalt array:

```
import numpy as np

arr1 = np.array([1, 2, 3, 4, 5])
print(arr1)           # Output: [1 2 3 4 5]
print(arr1[0])        # Forste element: 1
print(arr1.shape)     # Dimensjon: (5,)
```

To-dimensjonalt array:

```
arr2 = np.array([[1, 2, 3],
                 [4, 5, 6]])
print(arr2)
# Output:
# [[1 2 3]
#  [4 5 6]]

print(arr2[0,1])      # Element i forste rad, andre kolonne: 2
print(arr2.shape)     # Dimensjon: (2, 3)
print(arr2.flatten() # Output: array[1, 2, 3, 4, 5, 6]
```

Noen nyttige operasjoner:

- `arr2.flatten()` – gjør om 2D-array til 1D-array
- `arr2.shape` – viser dimensjonene

Enkle eksempler

```
import numpy as np

# Opprette en array
arr = np.array([1, 2, 3, 4])

# Ganger alle elementene med 2
print(arr * 2)
# Output: [2 4 6 8]

# Matriser
matrise = np.array([[1,2],[3,4]])
print(matrise)
```

```
# Output:
# [[1 2]
#  [3 4]]
```

Matematiske funksjoner i NumPy

NumPy har mange innebygde matematiske konstanter og funksjoner:

```
print(np.pi)            # Pi = 3.14159...
print(np.exp(1))        # e^1 = 2.718...
print(np.sqrt(16))      # Kvadratroten av 16 = 4
print(np.log(np.e))     # Naturlig logaritme av e = 1
```

Vektor- og matriseoperasjoner

I motsetning til vanlige lister støtter NumPy vektoriserte operasjoner (alle elementene behandles samtidig).

```
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])

print(a + b)  # [5 7 9]
print(a * b)  # [ 4 10 18]
```

Dette gjør at vi slipper løkker, og koden blir både raskere og enklere.

Økonomiske eksempler

NumPy er mye brukt i økonomi og finans, blant annet til beregninger med rente og nåverdi. Formelen for dette er

$$FV = PV \cdot (1 + r)^t$$

hvor:

- `FV` = Fremtidig verdi
- `PV` = Nåverdi
- `r` = rente
- `t` = antall perioder

```
# Eksempel: Beregning av fremtidig verdi (FV)
# FV = PV * (1 + r)^t
PV = 1000   # n verdi
r = 0.05    # rente (5 %)
t = 10      # antall perioder

FV = PV * (1 + r)**t
print(FV)  # Output: 1628.894626777442

# Flere verdier samtidig
renter = np.array([0.03, 0.04, 0.05])
FVs = PV * (1 + renter)**t
print(FVs)
# [1343.916 1480.244 1628.895]
```

Så her ser vi hva fremtidig verdi, blir for tre ulike renteverdier. Smart, ikke sant?

- NumPy kan raskt regne ut fremtidige verdier for mange ulike rentesatser.
- Brukes ofte til simuleringer, andre statistiske/matematiske beregninger.

Oppsummering – hvorfor NumPy?

- **Hastighet:** Raskere enn vanlige lister.
- **Funksjonalitet:** Har mange innebygde matematiske funksjoner og konstanter.
- **Vektor- og matriseoperasjoner:** Lar deg jobbe effektivt med store datasett.
- **Økonomiske analyser:** Svært nyttig for alt fra rente- og nåverdi-beregninger til statistikk og maskinlæring.

Del 1: Flervalgsoppgaver

1. Hva er hovedforskjellen på en liste og en tuple?

   a) Lister er mutable, tupler er immutable

   b) Tupler er raskere å iterere over

   c) Ingen forskjell

   d) Tupler kan bare inneholde tall

2. Hvilken metode bruker du for å legge til et element i en liste?

   a) `add()`

   b) `insert()`

   c) `append()`

   d) `extend()`

3. Hvordan definerer man et dictionary i Python?

   a) `[]`

   b) `()`

   c) `{}`

   d) `dict[]`

4. Hva vil følgende kode gi?

```
import numpy as np
arr = np.array([10,20,30])
print(arr.mean())

```

   a) 20

   b) 30

   c) Feil

   d) 60

5. Hva returnerer `len((1,2,3))`?

   a) 2

   b) 3

   c) (1,2,3)

   d) Feil

Del 2: Kortsvarsoppgaver

1. Hva blir output?

```
frukt = ["eple", "banan"]
frukt.append("appelsin")
print(frukt)

```

2. Hva blir resultatet?

```
koordinat = (4, 5)
print(koordinat[1])

```

3. Hvilken verdi får man her?

```
alder = {"Ola": 25, "Kari": 30}
print(alder["Kari"])

```

4. Hva vil dette gi?

```
import numpy as np
arr = np.array([2,4,6])
print(arr * 3)

```

5. Forklar med egne ord forskjellen på en liste og en tuple.

Del 3: Praktiske oppgaver

1. Lag en funksjon som tar en liste med tall som parameter og returnerer gjennomsnittet (uten NumPy).

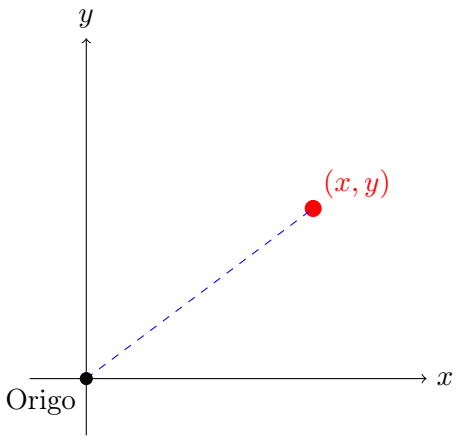
2. Lag en tuple som inneholder koordinatene til et punkt (x,y), og skriv en funksjon som returnerer avstanden til origo.

   Hint: Her bør du skrive noe pseudokode først! Og du kan få bruk for biblioteket `math`, og funksjonen `math.sqrt()`.

   Avstanden fra et punkt (x, y) til origo $(0, 0)$ kan finnes med Pytagoras-setningen:

   $$\text{avstand} = \sqrt{x^2 + y^2}$$

   Figur som illustrasjon:

3. Lag et dictionary som lagrer navn på studenter og poengsum. Lag en funksjon som finner studenten med høyest poengsum.

4. Bruk NumPy til å lage en array fra 1 til 10 og regn ut summen.

5. Lag en liste med tall, gjør den om til en NumPy-array og gang alle tallene med 2.

6. **Utfordring**: Lag et program som lagrer navn og favorittfarge i et dictionary. Programmet skal:

   (a) Spørre brukeren om et navn.

   (b) Sjekke om navnet finnes i dictionary.

   (c) Hvis navnet finnes, skriv ut favorittfargen.

   (d) Hvis navnet ikke finnes, be brukeren oppgi favorittfargen og legg den til i dictionary.

   Hint: Bruk `if/else`-setning, og `in` for å sjekke om nøkkelen finnes.

Fasit

Del 1: Flervalgsoppgaver

1:a, 2:c, 3:c, 4:a, 5:b

Del 2: Kortsvar

1: [eple", banan", "appelsin"]
2: 5
3: 30
4: [6 12 18]
5: Lister kan endres, tupler kan ikke endres.

Del 3: Praktiske oppgaver - Løsningsforslag

Oppgave 1

```python
def gjennomsnitt(liste):
    return sum(liste) / len(liste)
print(gjennomsnitt([2,4,6]))  # Output: 4.0
```

Gjennomsnittet finner vi med å legge sammen alle tallene i en liste (derfor: `sum(liste)`), delt på antall elementer i liste (derfor: `len(liste)`).

Oppgave 2

```python
import math # Importerer pakken math

def avstand(punkt):
    return math.sqrt(punkt[0]**2 + punkt[1]**2)

print(avstand((3,4)))  # Output: 5.0
```

Oppgave 3

```python
def topp_student(resultater):
    return max(resultater, key=resultater.get)

studenter = {"Ola": 80, "Kari": 95, "Per": 70}
print(topp_student(studenter))  # Output: Kari
```

- `resultater`, er parameteren til funksjonen `topp_student`, og er en **dictionary** med studentnavn som nøkler og poengsummer som verdier.
- `max(resultater, key=resultater.get)` finner nøkkelen med høyest verdi i dictionaryen.
- `print(best_student(studenter))` skriver ut navnet på studenten med høyest poengsum, fra dictionary'en `studenter`.

Oppgave 4

```python
import numpy as np
arr = np.arange(1,11)
print(arr.sum())
```

Oppgave 5

```
liste = [1,2,3,4]
arr = np.array(liste)
print(arr * 2)  # Output: [2 4 6 8]
```

Oppgave 6 - Utfordring

Løsningsforslag:

```
favorittfarger = {"Ola": "Gul", "Kari": "Oransj"}

navn = input("Skriv navn: ")
if navn in favorittfarger:
    print(f"Favorittfargen til {navn} er {favorittfarger[navn]}")
else:
    farge = input("Oppgi favorittfargen: ")
    favorittfarger[navn] = farge
    print(f"Fargen er lagt til!")
```