1 Ordbok og sett

Vi har sett de viktigste og generelle typene i Python i: `str`, `int`,`float`,`bool`,`list`. I denne seksjonen skal vi se på 2 andre typer samling som kan være nyttige i spesifikke situasjoner og som er vanlige typer i andre programmeringsspråk: `ordbok` og `sett`.

1.1 Ordbok

- En ordbok er en samling av nøkkel-verdi-par.
- Vi bruker syntaksen `{nøkkel1 : verdi1, nøkkel2 : verdi2, ...}` for å opprette en ordbok.
- Nøklene i en ordbok må være unike
- Verdiene i en ordbok kan være av forskjellige datatyper.
- Vi kan få tilgang til verdiene i en ordbok ved å referere til deres nøkkel , f. eks `min_ordbok["alder"]`.
- En ordbok har datatypen `dict` i Python.

[1]:
```
# En tom ordbok
tom_ordbok = {}
print(f"En tom ordbok: {tom_ordbok}")

# En ordbok med ulike typer informasjon om noe spesifikk
norge = {
    "land": "Norge",
    "befolkning" : 5500000,
    "språk" : ["bokmål", "nynorsk", "samisk"],
    "republikk" : False
}

# En ordbok med alltid samme type informasjon, men for flere ting.
hovedsteder = {
    "Norge" : "Oslo",
    "Danmark": "København",
    "Sverige" : "Stockholm",
}

print(f"Typen til en ordbok er:    {type(norge)}")

# Få tilgang til en verdi ved å bruke dens tilsvarende nøkkel.
print(f"Hovedstaden i Danmark er:  {hovedsteder['Danmark']}")

# Oppdater en verdi
hovedsteder["Norge"] = "Bergen"

# Opprett et nytt nøkkel-verdi par
hovedsteder["Frankrike"] = "Paris"

# Vi ser at ordboken ble oppdatert
print(hovedsteder)
```

```
En tom ordbok: {}
Typen til en ordbok er:    <class 'dict'>
Hovedstaden i Danmark er:  København
{'Norge': 'Bergen', 'Danmark': 'København', 'Sverige': 'Stockholm', 'Frankrike':
'Paris'}
```

1.1.1 Iterere over en ordbok

Vi ofte trenger å iterere over en ordbok. For å gjøre dette, bruker vi en `for`-løkke med ordbok metoden `items()`

Syntaks:

```
for (nøkkel, verdi) in ordbok.items():
    # kodeblokken med innrykk her
```

[2]:
```
for (land, hovedstad) in hovedsteder.items():
    print(f'Nøkkel: {land}    |    Verdi: {hovedstad}')
```

```
Nøkkel: Norge    |    Verdi: Bergen
Nøkkel: Danmark    |    Verdi: København
Nøkkel: Sverige    |    Verdi: Stockholm
Nøkkel: Frankrike    |    Verdi: Paris
```

1.1.2 Hvorfor en ordbok?

Man bruker en ordbok:

- Når man har behov for å lagre data i form av nøkkel-verdi-par.
- For å få en rask tilgang til data basert på unike nøkler.

En ordbok er ikke en sekvens, men en "mapping type"! Imidlertid er det flere sekvens metoder og operasjoner som er også implementerte for ordbøker.

1.1.3 Ordbok: flere metoder og operasjoner

Det forventes ikke at dere lærer alle disse metodene utenat! I cellen nedenfor gir vi dere bare en oversikt over hva som er mulig!

[3]:
```
norge = {
    "land": "Norge",
    "befolkning" : 5500000,
    "språk" : ["bokmål", "nynorsk", "samisk"],
    "republikk" : False
}
print(norge)

# Hvor mange nøkkel-verdi par er det?
print(f"{len(norge) = }")

# Sjekk om en *nøkkel* er i ordboka
print(f"{'språk' in norge = }")
print(f"{5500000 in norge = }")

# Ta ut en verdi fra ordbok
print(f'{norge.pop("språk")}')
# Nøkkelen språk er ikke lenger i ordboka (og den tilsvarende verdien)
print(norge)
# Gjør det samme men returnere verdien "Bergen" hvis nøkkelen
# "hovedstad" ikke er i ordboka
print(f'{norge.pop("hovedstad", "Bergen")}')
```

```
{'land': 'Norge', 'befolkning': 5500000, 'språk': ['bokmål', 'nynorsk',
'samisk'], 'republikk': False}
len(norge) = 4
'språk' in norge = True
5500000 in norge = False
['bokmål', 'nynorsk', 'samisk']
{'land': 'Norge', 'befolkning': 5500000, 'republikk': False}
Bergen
```

1.1.4 Ordbok: vanlige feil

Det forventes ikke at dere kjenner disse detaljene utenat! I cellen nedenfor prøver vi:

- å gjøre dere mer kjent til å lese feilmeldinger
- å gjøre dere mer kjent til å lese dokumentasjon / søke på internett hvis de oppstår en feil

[4]:
```python
# /!\ Det utløser en feilmelding /!\
# Bruken "pop" uten å spesifisere standardreturverdien utløser en feil
# om nøkkelen ikke er i ordboka!
print(f'{norge.pop("hovedstad")}')
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[4], line 4
      1 # /!\ Det utløser en feilmelding /!\
      2 # Bruken "pop" uten å spesifisere standardreturverdien utløser en feil
      3 # om nøkkelen ikke er i ordboka!
----> 4 print(f'{norge.pop("hovedstad")}')

KeyError: 'hovedstad'
```

1.2 Sett

- Et sett er en uordnet samling av unike elementer
- Vi bruker syntaksen `{element1, element2, ...}` for å opprette et sett.
- Vi kan opprette et sett fra en liste ved å bruke funksjonen `set()`
- Rekkefølgen som verdiene skrives ut eller nås samsvarer ikke med rekkefølgen ved opprettelse! Et sett en uordnet samling!
- Et sett har datatypen `set` i Python.

I eksempelet nedenfor:

- Vi har en liste over matrestriksjoner fra deltakerregistreringen
- Vi er bare interest i hva slags restriksjoner deltakerne har, ikke hvor mange personer har en gitt matrestriksjon

[5]:
```python
mitt_sett = {12, True, 12, "DIGI", 12, True}
# Et sett en uordnet samling!
print(mitt_sett)
print("Typen til et sett er: ", {type(mitt_sett)})

# En liste over matrestriksjoner fra deltakerregistreringen
```

```
mat_restriksjoner = [
    "egg", "nøtter", "kjøtt", "svinn", "fisk",
    "laktose", "gluten",  "fisk", "laktose", "gluten", "kjøtt",
    "nøtter", "melk", "svinn", "svinn", "egg",
    "laktose", "fisk", "melk", "egg", "egg", "melk", "svinn",
    "fisk", "nøtter", "nøtter", "melk", "fisk", "laktose",
    "nøtter", "gluten", "gluten", "kjøtt",
    "kjøtt"
]

# Vi kan opprette et sett fra en liste
unike_mat_restriksjoner = set(mat_restriksjoner)
print(unike_mat_restriksjoner)
```

```
{True, 'DIGI', 12}
Typen til et sett er:  {<class 'set'>}
{'fisk', 'egg', 'melk', 'gluten', 'laktose', 'nøtter', 'svinn', 'kjøtt'}
```

1.2.1 Hvorfor et sett?

Man bruker et sett:

- Når man trenger en samling av unike elementer uten en bestemt rekkefølge.
- For å eliminere duplikater fra en annen samling.
- For å få tilgang til matematiske settoperasjoner, som union, snitt og differanse.

1.2.2 Sett: flere metoder og operasjoner

Vi kan finne alle sett metoder i python her: https://docs.python.org/3/library/stdtypes.html#set-types-set-frozenset.

Det forventes ikke at dere lærer alle disse metodene utenat! I cellen nedenfor gir vi dere bare en oversikt over hva som er mulig!

[6]:
```
# Det er mulig og beregne hvor mange elementer det er i settet
len(unike_mat_restriksjoner)

# Det er mulig å iterere gjennom et sett
for element in mitt_sett:
    print(element)

# Og å sjekke om noe er i settet
print(f"{12 in mitt_sett = }")
```

```
True
DIGI
12
12 in mitt_sett = True
```

1.2.3 Et sett er ikke en sekvens!

Det er umulig å bruke indekser med sett!

[7]:
```
# /!\ Det utløser en feilmelding /!\
unike_mat_restriksjoner[0]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[7], line 2
      1 # /!\ Det utløser en feilmelding /!\
----> 2 unike_mat_restriksjoner[0]

TypeError: 'set' object is not subscriptable
```

1.2.4 Konvertering: funksjoner `list()` og `set()`

På samme måte som når vi konverterte heltall, flyttall og strenger (under visse forhold), kan vi konvertere lister og sett.

[8]:
```
print(mat_restriksjoner)

mitt_sett = set(mat_restriksjoner)
print(mitt_sett)

# Tilbake til en list men med bare unike verdier!
min_list = list(set(mat_restriksjoner))
print(min_list)
```

```
['egg', 'nøtter', 'kjøtt', 'svinn', 'fisk', 'laktose', 'gluten', 'fisk',
'laktose', 'gluten', 'kjøtt', 'nøtter', 'melk', 'svinn', 'svinn', 'egg',
'laktose', 'fisk', 'melk', 'egg', 'egg', 'melk', 'svinn', 'fisk', 'nøtter',
'nøtter', 'melk', 'fisk', 'laktose', 'nøtter', 'gluten', 'gluten', 'kjøtt',
'kjøtt']
{'fisk', 'egg', 'melk', 'gluten', 'laktose', 'nøtter', 'svinn', 'kjøtt'}
['fisk', 'egg', 'melk', 'gluten', 'laktose', 'nøtter', 'svinn', 'kjøtt']
```