

# **Dictionaries og strings**

**Ole Christian Lingjærde, Dept of Informatics, UiO**

4 oktober - 10 oktober 2021

# Denne ukens agenda

- Dictionaries
- Oppgave A.14 og 5.14 i Langtangens bok
- Tekststrenger
- Oppvarming til midtveiseksamen

**Dictionary:** en samling regler som knytter *nøkler* til *verdier*.

Eksempel:

```
d = {0:6, 1:3, 2:7}
```

Her er:

- Nøkler: 0, 1, 2
- Verdier: 6, 3, 7
- Regler:  $0 \rightarrow 6$ ,  $1 \rightarrow 3$ ,  $2 \rightarrow 7$

# Analogi til funksjoner

Tenk deg at du ønsker å implementere disse reglene:

```
"Norway" --> "Oslo"  
"Sweden" --> "Stockholm"  
"France" --> "Paris"
```

## Implementasjon med funksjon:

```
def f(x):  
    if x == "Norway":  
        return "Oslo"  
    elif x == "Sweden":  
        return "Stockholm"  
    elif x == "France":  
        return "Paris"
```

## Implementasjon med dictionary:

```
d = {"Norway": "Oslo", "Sweden": "Stockholm", "France": "Paris"}
```

# En dictionary kan utvides med nye regler

Tenk deg at vi i forrige eksempel ønsker en ny regel:

```
"Norway" --> "Oslo"  
"Sweden" --> "Stockholm"  
"France" --> "Paris"  
"Nepal" --> "Kathmandu" # NY REGEL
```

## Implementasjon med funksjon:

Må skrive en helt ny funksjon.

## Implementasjon med dictionary:

Kan enkelt utvide en dictionary med nye regler:

```
d["Nepal"] = "Kathmandu"
```

Nøkler i en dictionary = indekser i en liste

Merk: indeksene er alltid 0, 1, ..., N-1 i lister, mens nøklene kan være nesten hva som helst i en dictionary.

Eksempler:

```
# Parene (-2,6), (6,3), (3,7)
d = {-2:6, 6:3, 3:7}
```

```
# Parene ("Hamar", "Norway"), ("Uppsala", "Sweden")
d = {"Hamar": "Norway", "Uppsala": "Sweden"}
```

```
# Parene ("pi", 3.14), ("e", 2.718), ("g", 9.81)
d = {"pi": 3.14, "e": 2.718, "g": 9.81}
```

```
# Parene ((0,0), "lowerleft") and ((1,1), "upperright")
d = {(0,0): "lowerleft", (1,1): "upperright"}
```

# Nøklene i en dictionary må være "immutable"

Nøklene i en dictionary må være *immutable*, dvs objekter som ikke kan endres.

Eksempler:

- int-verdier og float-verdier er immutable
- string-verdier er immutable
- tupler er immutable
- lister er *ikke* immutable (kan endre element med `a[i] = y`)
- objekter av egendefinerte klasser er *ikke* immutable

## To regler kan ikke ha samme nøkkel:

Vi kan ikke skrive  $d = \{\text{"Oslo"}: 3, \text{"Oslo"}: 4\}$ . Hvis vi definerer  $d = \{\text{"Oslo"}: 3\}$  og så skriver  $d[\text{"Oslo"}] = 5$  så overskrives den første regelen.

## To regler kan ha samme verdi:

Vi kan godt skrive  $d = \{\text{"Oslo"}: 3, \text{"Bergen"}: 3\}$ .

## Reglene har ingen ordning:

En dictionary lagrer ikke reglene i en bestemt rekkefølge. Tenk på en dictionary som en "sekk med regler".



Hvilke av disse forsøkene på å lage en dictionary er korrekte, og hva er isåfall nøklene og verdiene?

- `d = {3;8}`
- `d = [3:8]`
- `d = {3:8}`
- `d = {3:8, 3:5}`
- `d = {8:3, 5:3}`
- `d = { ("A", "A"): "homozygot", ("A", "B"): "heterozygot" }`
- `d = { ["A", "A"]: "homozygot", ["A", "B"]: "heterozygot" }`

## Initialisere en dictionary:

```
# Lag en tom dictionary:  
d = {}  
# Lag en dictionary med to par:  
d = {"Oslo": 13, "London": 15.4}  
# Lag en dictionary med to par (alternativ metode):  
d = dict(Oslo=13, London=15.4)
```

## Utvide en dictionary:

```
# Anta at d er en dictionary  
# Legg et nytt par til d:  
d["Madrid"] = 26.0  
# Legg en hel dictionary d2 til d:  
d.update(d2)
```

# Å fjerne regler fra en dictionary

## Fjerne et par (a,b) fra en dictionary:

```
d = {"pi":3.14, "e":2.718, "g":9.81}
del d["e"]           # Fjern paret ("e", 2.718)
del d["pi"]         # Fjern paret ("pi", 3.14)
```

**Merk:** forsøk på å fjerne en nøkkel som ikke finnes i en dictionary gir feilmelding.

# Teste om en nøkkel finnes i en dictionary

## **key in d:**

Tester om `key` er en nøkkel i `d`

## **d[key]:**

Henter ut verdien i `d` lagret under nøkkelen `key`. Gir feilmelding hvis `key` ikke er en nøkkel i `d`.

## **d.get(key):**

Henter ut verdien i `d` lagret under nøkkelen `key`. Returnerer `None` hvis `key` ikke er en nøkkel i `d`.

# Eksempel

```
d = {"Berkeley": "US", "Cambridge": "UK"}  
key = "Berkeley"
```

## *# Alternativ A:*

```
if key in d:  
    print(f"Regelen {key} --> {d[key]} ligger i dictionary")  
else:  
    print(f"Ingen regel med nøkkel {key} i dictionary")
```

## *# Alternativ B:*

```
value = d.get(key)  
  
if value != None:  
    print(f"Regelen {key} --> {d[key]} ligger i dictionary")  
else:  
    print(f"Ingen regel med nøkkel {key} i dictionary")
```

## Løpe over elementene i **vilkårlig** rekkefølge:

```
d = {-2:6, 6:3, 3:7}
for key in d:
    print("Key = %g and value = %g" % (key, d[key]))
```

## Løpe over elementene i **sortert** nøkkelrekkefølge:

```
d = {-2:6, 6:3, 3:7}
for key in sorted(d):
    print("Key = %g and value = %g" % (key, d[key]))
```

# Lage liste over alle nøkler/verdier

## Anta at vi har definert en dictionary:

```
d = {"Paris": 17.5, "London": 15.4, "Madrid": 26.0}
```

```
> d.keys()
```

```
<listiterator at 0x111b2a4d0>
```

```
> list(d.keys())
```

```
["Paris", "London", "Madrid"]
```

```
> d.values()
```

```
<listiterator at 0x111b2a710>
```

```
> list(d.values()) [17.5,
```

```
15.4, 26.0]
```

# Eksempel: Lese to-kolonne fil til dictionary

## Datafil:

MB.0000	0.00096
MB.0002	0.24787
MB.0005	0.2779
MB.0006	0.29428
MB.0010	0.61225
...	...



# Eksempel: Lese to-kolonne fil til dictionary

## Datafil:

MB.0000	0.00096
MB.0002	0.24787
MB.0005	0.2779
MB.0006	0.29428
MB.0010	0.61225
...	...

Første oppgave: å bestemme hva som skal være nøkkel og hva som skal være verdi.

# Eksempel: Lese to-kolonne fil til dictionary

## Datafil:

MB.0000	0.00096
MB.0002	0.24787
MB.0005	0.2779
MB.0006	0.29428
MB.0010	0.61225
...	...

**Første oppgave:** å bestemme hva som skal være nøkkel og hva som skal være verdi.

**Nøkkel:** første kolonne (f.eks. MB.0000)

**Verdi:** andre kolonne (f.eks. 0.00096)

# Eksempel: Lese to-kolonne fil til dictionary

## Datafil:

```
MB.0000      0.00096
MB.0002      0.24787
MB.0005      0.2779
MB.0006      0.29428
MB.0010      0.61225
...          ...
```

## Program:

```
bloodtest = {}
infile = open("blood_test.txt", "r")
for line in infile:
    words = line.split()
    bloodtest[words[0]] = float(words[1])
infile.close()
# Skrive ut verdien til nøkkel MB.0005:
print(bloodtest["MB.0005"])          #
0.00096
```

# Eksempel: Lese tre-kolonne fil til dictionary

## Datafil:

Oslo	21.8	"Norway"
Bergen	17.6	"Norway"
London	18.1	"UK"
Berlin	19	"Germany"
Paris	23	"France"
Rome	26	"Italy"
Helsinki	17.8	"Finland"

# Eksempel: Lese tre-kolonne fil til dictionary

## Datafil:

Oslo	21.8	"Norway"
Bergen	17.6	"Norway"
London	18.1	"UK"
Berlin	19	"Germany"
Paris	23	"France"
Rome	26	"Italy"
Helsinki	17.8	"Finland"

**Første oppgave:** å bestemme hva som skal være nøkkel og hva som skal være verdi.

**Nøkkel:** første kolonne (f.eks. MB.0000)

**Verdi:** andre og tredje kolonne (som liste eller tuppel)

# Eksempel: Lese tre-kolonne fil til dictionary

## Datafil:

Oslo	21.8	"Norway"
Bergen	17.6	"Norway"
London	18.1	"UK"
Berlin	19	"Germany"
Paris	23	"France"
Rome	26	"Italy"
Helsinki	17.8	"Finland"

## Program:

```
infile = open("cityinfo.txt", "r")
data = {}
for line in infile:
    words = line.split()
    data[words[0]] = [float(words[1]), words[2]]
infile.close()
```

```
# Skrive ut informasjonen til nøkkel Paris:
print(data["Paris"]) # [23.0, "France"]
print(data["Paris"][0]) # 23.0
print(data["Paris"][1]) # "France"
```

# Eksempel: Lese fil med navngitte kolonner

Datafilen `table.dat`:

Navn	Alder	Utdanningssted
Anne	10	Eiksmarka
Tom	15	Marienlyst
Vidar	18	Persbråten
Johanne	17	Wang
Silje	16	Eikeli
Per	19	UiO

# Eksempel: Lese fil med navngitte kolonner

Datafilen `table.dat`:

Navn	Alder	Utdanningssted
Anne	10	Eiksmarka
Tom	15	Marienlyst
Vidar	18	Persbråten
Johanne	17	Wang
Silje	16	Eikeli
Per	19	UiO

Løsningsskisse:

```
infile = open("table.dat", "r")
data = {}
headers = infile.readline().split()
for i in range(len(headers)):
    data[headers[i]] = []
for line in infile:
    words = line.split()
    for i in range(len(headers)):
        data[headers[i]].append(words[i])
infile.close()
```

Les overskrifter

Les data



# Quiz 1

Hva skrives ut?

Spørsmål A:

```
d = {-2:-1, -1:0, 0:1, 1:2, 2:-2}
print(d[0])
```

Spørsmål B:

```
d = {-2:-1, -1:0, 0:1, 1:2, 2:-2}
print(d[d[0]])
```

Spørsmål C:

```
d = {-2:-1, -1:0, 0:1, 1:2, 2:-2}
print(d[-2]*d[2])
```

Hva skrives ut?

Spørsmål A:

```
d = {-2:-1, -1:0, 0:1, 1:2, 2:-2}
print(d[0])           # 1
```

Spørsmål B:

```
d = {-2:-1, -1:0, 0:1, 1:2, 2:-2}
print(d[d[0]])       # 2
```

Spørsmål C:

```
d = {-2:-1, -1:0, 0:1, 1:2, 2:-2}
print(d[-2]*d[2])    # 2
```

## Quiz 2

Hva skrives ut?

Spørsmål A:

```
table = {"age": [35, 20], "name": ["Anna", "Peter"]}
for key in table:
    print("%s: %s" % (key, table[key]))
#
#
```

Spørsmål B:

```
table = {"age": [35, 20], "name": ["Anna", "Peter"]}
vals = list(table.values())
print(vals)
print(vals[0])
print(vals[0][0])
#
#
#
```

Spørsmål C:

```
table = {"age": [35, 20], "name": ["Anna", "Peter"]}
print(table["name"][1], table["age"][1])
#
```

# Svar på Quiz 2

Hva skrives ut?

Spørsmål A:

```
table = {"age": [35, 20], "name": ["Anna", "Peter"]}
for key in table:
    print("%s: %s" % (key, table[key]))
# age: [35, 20]
# name: ["Anna", "Peter"]
```

Spørsmål B:

```
table = {"age": [35, 20], "name": ["Anna", "Peter"]}
vals = list(table.values())
print(vals)
print(vals[0])
print(vals[0][0])
# [[35, 20], ["Anna", "Peter"]]
# [35, 20]
# 35
```

Spørsmål C:

```
table = {"age": [35, 20], "name": ["Anna", "Peter"]}
print(table["name"][1], table["age"][1])
# ("Peter", 20)
```

Hva blir innholdet i oppslagstabellen d?

**Spørsmål A:**

d = {3:5, 6:7}

e = {4:6, 7:8}

d.update(e)

**Spørsmål B:**

d = {3:5, 6:7}

e = {4:6, 7:8}

d.update(e)

d.update(e)

**Spørsmål C:**

d = {6:100}

e = {6:6, 7:8}

d.update(e)

# Svar på Quiz 3

Hva blir innholdet i oppslagstabellen d?

Spørsmål A:

```
d = {3:5, 6:7}
e = {4:6, 7:8}
d.update(e)
# {3: 5, 4: 6, 6: 7, 7: 8}
```

Spørsmål B:

```
d = {3:5, 6:7}
e = {4:6, 7:8}
d.update(e)
d.update(e)
# {3: 5, 4: 6, 6: 7, 7: 8}
```

Spørsmål C:

```
d = {6:100}
e = {6:6, 7:8}
d.update(e)
# {6: 6, 7: 8}
```

Filen "teledata.txt" inneholder info om mobilkunder:

Age	Income	Gender	Monthly calls	ID
45	720k	Female	46	A001
27	440k	Male	3	A002
17	0	Male	52	A006
24	60k	Female	18	A014
...	...	...	...	...

Kom med forslag til hvordan vi kan:

- lagre dataene som fem lister
- lagre dataene som én liste
- lagre dataene som en dictionary

## Svar på Quiz 4

```
# Fem lister:  
# ["Age", 45, 27, ...]  
# ["Income", "720k", "440k", ...]  
# ["Gender", "Female", "Male", ...]  
# ["Monthly calls", 46, 3, ...]  
# ["ID", "A001", "A002", ...]  
  
# En liste:  
# [{"Age":45,27,17,...}, {"Income":"720k", "440k", ...}], osv]  
  
# En oppslagstabell:  
# {"Age":[45,27,17,...], "Income":["720k,440k,0,..."], osv}
```



**Exercise A.14: Find difference equations for computing  $\sin x$** 

The purpose of this exercise is to derive and implement difference equations for computing a Taylor polynomial approximation to  $\sin x$ :

$$\sin x \approx S(x; n) = \sum_{j=0}^n (-1)^j \frac{x^{2j+1}}{(2j+1)!}. \quad (\text{A.60})$$

To compute  $S(x; n)$  efficiently, write the sum as  $S(x; n) = \sum_{j=0}^n a_j$ , and derive a relation between two consecutive terms in the series:

$$a_j = -\frac{x^2}{(2j+1)2j} a_{j-1}. \quad (\text{A.61})$$

Introduce  $s_j = S(x; j-1)$  and  $a_j$  as the two sequences to compute. We have  $s_0 = 0$  and  $a_0 = x$ .

a) Formulate the two difference equations for  $s_j$  and  $a_j$ .

## Exercise A.14

Taylor approksimasjon til  $\sin x$ :

$$\sin x \approx \sum_{j=0}^n (-1)^j \frac{x^{2j+1}}{(2j+1)!}$$

For å regne ut det  $j$ te leddet må vi regne ut

- $x^{2j+1}$
- $(2j+1)!$
- $(-1)^j$
- ... og så multiplisere og dividere

Kan vi gjøre det mer effektivt?

**Hint:** Det er lett å komme fra ledd  $j-1$  til ledd  $j$ .

## Exercise A.14

Ledd  $j - 1$ :

$$a_{j-1} = (-1)^{j-1} \frac{x^{2(j-1)+1}}{(2(j-1)+1)!} = (-1)^{j-1} \frac{x^{2j-1}}{(2j-1)!}$$

Ledd  $j$ :

$$a_j = (-1)^j \frac{x^{2j+1}}{(2j+1)!} = -\frac{x^2}{2j(2j+1)} \cdot a_{j-1}$$

Nå trengs bare tre multiplikasjoner + en divisjon + et fortegnsskift.

## Exercise A.14

Oppsummert har vi nå at

$$\sin x \approx \sum_{j=0}^n a_j$$

hvor

$$a_j = -\frac{x^2}{2j(2j+1)} \cdot a_{j-1} \quad j = 1, 2, 3, \dots$$

I tillegg må vi sette initialbetingelsen  $a_0 = x$ .

## Exercise A.14

Vi kan skrive Taylor-approximasjonen  $\sin x \approx \sum_{j=0}^n a_j$  på en litt annen måte:

$$\sin x \approx s_{n+1}$$

hvor vi har for  $n=1,2,3,\dots$

$$\begin{aligned} s_n &= s_{n-1} + a_{n-1} \\ a_n &= -\frac{x^2}{2n(2n+1)} \cdot a_{n-1} \end{aligned}$$

og hvor  $s_0=0$  og  $a_0 = x$ .

- b) Implement the system of difference equations in a function `sin_Taylor(x, n)`, which returns  $s_{n+1}$  and  $|a_{n+1}|$ . The latter is the first neglected term in the sum (since  $s_{n+1} = \sum_{j=0}^n a_j$ ) and may act as a rough measure of the size of the error in the Taylor polynomial approximation.
- c) Verify the implementation by computing the difference equations for  $n = 2$  by hand (or in a separate program) and comparing with the output from the `sin_Taylor` function. Automate this comparison in a test function.
- d) Make a table or plot of  $s_n$  for various  $x$  and  $n$  values to illustrate that the accuracy of a Taylor polynomial (around  $x = 0$ ) improves as  $n$  increases and  $x$  decreases.

*Hint* Be aware of the fact that `sin_Taylor(x, n)` can give extremely inaccurate approximations to  $\sin x$  if  $x$  is not sufficiently small and  $n$  sufficiently large. In a plot you must therefore define the axis appropriately.

Filename: `sin_Taylor_series_diffeq`.

### Exercise 5.14: Plot data in a two-column file

The file `src/plot/xy.dat`<sup>12</sup> contains two columns of numbers, corresponding to  $x$  and  $y$  coordinates on a curve. The start of the file looks as this:

```
-1.0000    -0.0000
-0.9933    -0.0087
-0.9867    -0.0179
-0.9800    -0.0274
-0.9733    -0.0374
```

Make a program that reads the first column into a list `x` and the second column into a list `y`. Plot the curve. Print out the mean  $y$  value as well as the maximum and minimum  $y$  values.

*Hint* Read the file line by line, split each line into words, convert to `float`, and append to `x` and `y`. The computations with `y` are simpler if the list is converted to an array.

Filename: `read_2columns`.