

Dictionaries og strings

Ole Christian Lingjærde, Dept of Informatics, UiO

28 september - 2 oktober 2020

Denne ukens agenda

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

Dictionary: 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: $0 \mapsto 6, 1 \mapsto 3, 2 \mapsto 7$

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"
```

Analogi til lister

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 "konstante"

Nøklene i en dictionary må være konstante (*immutable*) objekter, dvs objekter som ikke går an å endre.

Eksempler:

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

To regler kan ikke ha samme nøkkel:

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

To regler kan ha samme verdi:

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

Reglene har ingen ordning:

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

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:

Vi kan teste om en nøkkel finnes i en dictionary med `key in d`.

d[key]:

Vi kan hente ut en verdi med `verdi = d[key]`. Vi får feilmelding hvis nøkkelen ikke finnes i dictionaryen.

d.get(key):

Vi kan også hente ut en verdi med `verdi = d.get(key)`. Verdien `None` returneres hvis nøkkelen ikke finnes i dictionaryen.

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")
```

Å løpe gjennom elementene i en 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]))
```

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
...          ...
```

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"

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"
```

Lese en 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

Løsningskisse:

```
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()
```

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])
```

Svar på Quiz 1

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,...}, {"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 $\sin x$:

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

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:

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

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 .

Hint: Section A.1.8 explains how this task can be solved for the Taylor approximation of e^x .

Exercise A.14 (cont'd)

- 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: `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.

Key idea

Computing series with many terms can be time-consuming. A common strategy is to see if the n th term can be found faster using the $(n - 1)$ st term.

Calculating $S(x; n) = a_0 + a_1 + \cdots + a_n$

Suppose in the following that x is fixed and let $s_{n+1} = S(x; n)$.

1) We can find s_{n+1} from s_n and a_n :

$$s_{n+1} = s_n + a_n$$

2) We can also find a_n using a_{n-1} and x :

$$a_{n-1} = (-1)^{n-1} \frac{x^{2n-1}}{(2n-1)!} \text{ and } a_n = (-1)^n \frac{x^{2n+1}}{(2n+1)!}$$

Thus we have the relation:

$$a_n = -a_{n-1} \cdot \frac{x^2}{2n(2n+1)}$$

Answer to exercise A.14 a)

Question

Formulate the two difference equations for s_n and a_n .

Answer

Set $s_0 = 0$ and $a_0 = x$. For $n = 1, 2, \dots$ let

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

Answer to exercise A.14 b)

Question

Implement the system of difference equations in a function `sin_Taylor(x, n)` which returns s_{n+1} and $|a_{n+1}|$.

Answer 1: storing all updates

```
def sin_Taylor(x,n):
    s = [0.0]*(n+2)
    a = [0.0]*(n+2); a[0] = x
    for i in range(1,n+2):
        s[i] = s[i-1] + a[i-1]
        a[i] = -a[i-1]*x**2/(2*i*(2*i+1))
    return s[n+1], abs(a[n+1])
```

Answer 2: storing only last update

```
def sin_Taylor2(x,n):
    s = 0
    a = x
    for i in range(1,n+2):
        s = s + a
        a = -a*x**2/(2*i*(2*i+1))
    return s, abs(a)
```

Answer to exercise A.14 c)

Question

Verify the implementation by computing the difference equations for $n = 2$ and comparing with the output from the `sin_Taylor` function. Automate this comparison in a test function.

Answer (part 1)

```
def sin_two_terms(x):
    s = [0]*4
    a = [0]*4

    a[0] = x

    s[1] = s[0] + a[0]
    a[1] = -a[0]*x**2 / (2*1*(2*1+1))

    s[2] = s[1] + a[1]
    a[2] = -a[1]*x**2 / (2*2*(2*2+1))

    s[3] = s[2] + a[2]
    a[3] = -a[2]*x**2 / (2*3*(2*3+1))

    return s[3], abs(a[3])
```

Answer to exercise A.14 c)

Answer (part 2)

```
def sin_Taylor(x):  
    <as before>  
  
def sin_two_terms(x):  
    <as before>  
  
def test_sin_Taylor():  
    x = 0.63      # Just an arbitrary x-value for validation  
    tol = 1e-14  # Tolerance  
    s_expected, a_expected = sin_two_terms(x)  
    s_computed, a_computed = sin_Taylor(x,2)  
    success1 = abs(s_computed - s_expected) < tol  
    success2 = abs(a_computed - a_expected) < tol  
    success = success1 and success2  
    message = 'Output is different from expected!'  
    assert success, message
```

Answer to exercise A.14 c)

Answer (part 3)

```
In [10]: sin_two_terms(0.63)
```

```
Out[10]: (0.5891525304525, 7.815437776125003e-06)
```

```
In [11]: sin_Taylor(0.63, 2)
```

```
Out[11]: (0.5891525304525, 7.815437776125003e-06)
```

```
In [12]: test_sin_Taylor()
```

```
In [13]:
```

Answer to exercise A.14 d)

Question

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.

Answer (part 1)

We first make a plan:

- For a given x and n we can calculate the Taylor approximation s_{n+1} with the statement `s = sin_Taylor(x,n) [0]`.
- To calculate s_n for various x and n , we must decide what x -values and n -values to use.
- We decide here to use a uniform grid of M x -values on $[0, 1]$, and $n = 1, 2, \dots, N$.
- We write a method producing an $M \times N$ table of all the calculated s -values.

Answer to exercise A.14 d)

Answer (part 2)

```
def make_table(M,N):
    import numpy as np
    x = np.linspace(0, 1, M)
    n = np.arange(1,N+1)
    S = np.zeros((M,N))
    for i in range(M):
        for j in range(N):
            S[i,j] = sin_Taylor(x[i], n[j])[0]
    return S, x, n
```

Exercise 5.14

Plot data in a two-column file

The file:

`https://github.com/hplgit/scipro-primer/blob/master/src/plot/xy.dat`

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.`

Answer to exercise 5.14

```
# Read file
infile = open('xy.dat12', 'r')
x = []
y = []
for line in infile:
    s = line.split()
    x.append(eval(s[0]))
    y.append(eval(s[1]))
infile.close()

# Plot the points (x[i],y[i])
import matplotlib.pyplot as plt
plt.plot(x, y, 'mo')
plt.xlabel('x')
plt.ylabel('y')
plt.show()

# Print mean, min and max of y
import numpy as np
ya = np.array(y)
print('Mean of y: %g' % np.mean(ya))
print('Min of y: %g' % np.min(ya))
print('Max of y: %g' % np.max(ya))
```

Result

