Lær Python dag 2 - modul 2

Strenge, dictionaries

Jonas Bamse Andersen

Institut for Matematik og Datalogi - IMADA
Syddansk Universitet

Indhold

1. Strenge
2. Dictionaries

Strenge

Disclaimer

Jeg forsøger at give et overblik over nogle muligheder, men forventer ikke I husker det hele.

Strenge - en sekvens af tegn

I har set tekststrenge mange gange allerede.

```
name = input("Intast dit navn: ")
print("Hej " + name + "!")
```

Strenge - en sekvens af tegn

Men strenge kan bruges til alt muligt, f.eks. til at repræsentere en DNA-sekvens eller en bog.

```
dna = "ATTAGCC"
book = "Once upon a time ..."
```

Strenge - en sekvens af tegn

En streng er på mange måder bare en liste af enkelte tegn.

```
dna = "ATTAGCC"
```

Tegn	A	T	T	A	G	C	C
Index	0	1	2	3	4	5	6

Strenge - en sekvens af tegn

En streng er på mange måder bare en liste af enkelte tegn.

```
dna = "ATTAGCC"
```

Tegn	A	T	T	A	G	C	C
Index	0	1	2	3	4	5	6

Så derfor kan vi bruge mange af de samme funktioner som vi lærte til lister!

Strenge - en sekvens af tegn

Indeksering

```
dna = "ATTAGCC"
print(dna[1])
```

Strenge - en sekvens af tegn

Indeksering

```
dna = "ATTAGCC"
print(dna[1])
```

```
T
```

Strenge - en sekvens af tegn

Indeksering

```
dna = "ATTAGCC"
print(dna[1])
```

```
T
```

Slicing

```
dna = "ATTAGCC"
print(dna[2:5])
```

Strenge - en sekvens af tegn

Indeksering

```
dna = "ATTAGCC"
print(dna[1])
```

```
T
```

Slicing

```
dna = "ATTAGCC"
print(dna[2:5])
```

```
TAG
```

Strenge - en sekvens af tegn

Indeksering

```
dna = "ATTAGCC"
print(dna[1])
```

```
T
```

Slicing

```
dna = "ATTAGCC"
print(dna[2:5])
```

```
TAG
```

Længde

```
dna = "ATTAGCC"
print(len(dna))
```

Strenge - en sekvens af tegn

Indeksering

```
dna = "ATTAGCC"
print(dna[1])
```

```
T
```

Slicing

```
dna = "ATTAGCC"
print(dna[2:5])
```

```
TAG
```

Længde

```
dna = "ATTAGCC"
print(len(dna))
```

```
7
```

Strenge - en sekvens af tegn

Gennemløb af liste (iterate)

```
dna = "ATTAGCC"
for c in dna:
    print(c)
```

Strenge - en sekvens af tegn

Gennemløb af liste (iterate)

```
dna = "ATTAGCC"
for c in dna:
    print(c)
```

```
A
T
T
A
G
C
C
```

Strenge - en sekvens af tegn

Lister er mutable, hvad med strenge?

```
mylist = [1, 2, 3, 4]
mylist[0] = 5
print(mylist)

dna = "ATTAGCC"
dna[0] = "G"
print(dna)
```

Strenge - en sekvens af tegn

Lister er mutable, hvad med strenge?

```
mylist = [1, 2, 3, 4]
mylist[0] = 5
print(mylist)

dna = "ATTAGCC"
dna[0] = "G"
print(dna)
```

```
[5, 2, 3, 4]
Traceback (most recent
    call last):
  File "test.py", line
      6, in <module>
    dna[0] = "G"
TypeError: 'str' object
    does not support
    item assignment
```

Strenge - en sekvens af tegn

En løsning på immutability er at bygge nye strenge

```
dna = "ATTAGCC"
newdna = "G" + dna[1:]
print(newdna)
```

Strenge - en sekvens af tegn

En løsning på immutability er at bygge nye strenge

```
dna = "ATTAGCC"
newdna = "G" + dna[1:]
print(newdna)
```

```
GTTAGCC
```

Strenge - en sekvens af tegn

Hvad med `.append()`?

```
dna = "ATTAGCC"
dna.append("A")
print(dna)
```

Strenge - en sekvens af tegn

Hvad med `.append()`?

```
dna = "ATTAGCC"
dna.append("A")
print(dna)
```

```
Traceback (most recent
    call last):
  File "test.py", line
    2, in <module>
    dna.append("A")
AttributeError: 'str'
    object has no
    attribute 'append'
```

Strenge - en sekvens af tegn

Heldigvis ved vi at vi kan sammensætte (konkatenere) strenge med "+"

```
dna = "ATTAGCC"
newdna = dna + "A"
print(newdna)
```

Strenge - en sekvens af tegn

Heldigvis ved vi at vi kan sammensætte (konkatenere) strenge med "+"

```
dna = "ATTAGCC"
newdna = dna + "A"
print(newdna)
```

```
ATTAGCCA
```

Strenge - en sekvens af tegn

Der er også visse funktioner (metoder) som er lavet til strenge, f.eks. `.lower()` og `.upper()` som ændrer alle bogstaver til henholdsvis små og store.

```
dna = "ATTAGCC"
newdna = dna.lower()
print(newdna)
```

Strenge - en sekvens af tegn

Der er også visse funktioner (metoder) som er lavet til strenge, f.eks. `.lower()` og `.upper()` som ændrer alle bogstaver til henholdsvis små og store.

```
dna = "ATTAGCC"
newdna = dna.lower()
print(newdna)
```

```
attagcc
```

Strenge - en sekvens af tegn

Hvis man er interesseret i at vide hvor i en streng et tegn forekommer, kan man bruge `.find()`.

```
dna = "ATTAGCC"
print(dna.find("G"))
```

Strenge - en sekvens af tegn

Hvis man er interesseret i at vide hvor i en streng et tegn forekommer, kan man bruge `.find()`.

```
dna = "ATTAGCC"
print(dna.find("G"))
```

```
4
```

Strenge - en sekvens af tegn

Hvis man er interesseret i at vide hvor i en streng et tegn forekommer, kan man bruge `.find()`.

```
dna = "ATTAGCC"
print(dna.find("G"))
```

```
4
```

Hvad hvis der er flere af den samme, eller slet ikke nogen?

```
dna = "ATTAGCC"
print(dna.find("A"))
print(dna.find("B"))
```

Strenge - en sekvens af tegn

Hvis man er interesseret i at vide hvor i en streng et tegn forekommer, kan man bruge `.find()`.

```
dna = "ATTAGCC"
print(dna.find("G"))
```

```
4
```

Hvad hvis der er flere af den samme, eller slet ikke nogen?

```
dna = "ATTAGCC"
print(dna.find("A"))
print(dna.find("B"))
```

```
0
-1
```

Første forekomst og -1

Strenge - en sekvens af tegn

Kan man søge på mere end bare et enkelt tegn?

```
dna = "ATTAGCC"
print(dna.find("TAG"))
```

Strenge - en sekvens af tegn

Kan man søge på mere end bare et enkelt tegn?

```
dna = "ATTAGCC"
print(dna.find("TAG"))
```

```
2
```

Ja, svaret er index fra,
hvor forekomsten starter.

Strenge - en sekvens af tegn

Hvis man er ligeglad med HVOR en forekomst er, men bare vil vide OM det forekommer kan man bruge `in`.

```
dna = "ATTAGCC"
print("TAG" in dna)
print("CAT" in dna)
```

Strenge - en sekvens af tegn

Hvis man er ligeglad med HVOR en forekomst er, men bare vil vide OM det forekommer kan man bruge `in`.

```
dna = "ATTAGCC"
print("TAG" in dna)
print("CAT" in dna)
```

```
True
False
```

Strenge - en sekvens af tegn

Man kan inddele en sætning i ord ved at skære den i stykker ved mellemrum og få alle stykkerne i en liste. Til det bruger man `.split()` funktionen.

Strenge - en sekvens af tegn

Man kan inddele en sætning i ord ved at skære den i stykker ved mellemrum og få alle stykkerne i en liste. Til det bruger man `.split()` funktionen.

Man skal fortælle ved hvilket slags tegn man gerne vil skære (ofte mellemrum).

```
s = "Jeg gik mig over sø
    og land"
l = s.split(" ")
print(l)
print(l[2])
```

Strenge - en sekvens af tegn

Man kan inddele en sætning i ord ved at skære den i stykker ved mellemrum og få alle stykkerne i en liste. Til det bruger man `.split()` funktionen.

Man skal fortælle ved hvilket slags tegn man gerne vil skære (ofte mellemrum).

```
s = "Jeg gik mig over sø
    og land"
l = s.split(" ")
print(l)
print(l[2])
```

```
['Jeg', 'gik', 'mig', '
    over', 'sø', 'og', '
    land']
mig
```

Strenge - en sekvens af tegn

Det modsatte af at splitte er at sætte sammen, til det bruger man .join(). Den sætter en streng ind mellem alle elementerne i listen. Det bruges ofte til hurtigt at lave en liste af strenge til en enkelt streng, ved at joine med mellemrum.

```
l = ['Jeg', 'gik', 'mig'
    , 'over', 'sø', 'og',
     'land']
binde = "-".join(l)
print(binde)
```

Strenge - en sekvens af tegn

Det modsatte af at splitte er at sætte sammen, til det bruger man .join(). Den sætter en streng ind mellem alle elementerne i listen. Det bruges ofte til hurtigt at lave en liste af strenge til en enkelt streng, ved at joine med mellemrum.

```
l = ['Jeg', 'gik', 'mig'
     , 'over', 'sø', 'og',
      'land']
binde = "-".join(l)
print(binde)
```

```
Jeg-gik-mig-over-sø-
    og-land
```

Strenge - en sekvens af tegn

Man kan også erstatte bogstaver med `.replace()`. Her skal man fortælle, hvad der skal ændres og hvad det skal ændres til.

Strenge - en sekvens af tegn

Man kan også erstatte bogstaver med `.replace()`. Her skal man fortælle, hvad der skal ændres og hvad det skal ændres til.

Kan f.eks. bruges til at fjerne mellemrum.

```
s = "Jeg gik mig over sø
    og land"
s2 = s.replace(" ", "")
     # Erstatter
    mellemrum med
    ingenting =
    fjerner mellemrum
print(s2)
```

Strenge - en sekvens af tegn

Man kan også erstatte bogstaver med `.replace()`. Her skal man fortælle, hvad der skal ændres og hvad det skal ændres til.

Kan f.eks. bruges til at fjerne mellemrum.

```
s = "Jeg gik mig over sø
    og land"
s2 = s.replace(" ", "")
     # Erstatter
    mellemrum med
    ingenting =
    fjerner mellemrum
print(s2)
```

```
Jeggikmigoversøogland
```

Strenge - en sekvens af tegn

Et eksempel på brug af nogle af de ting vi nu har lært kunne være en funktion som givet to ord (strenge), printer alle bogstaver der forekommer i begge ord.

```
def in_both(word1, word2):
  #code?
```

Strenge - en sekvens af tegn

Et eksempel på brug af nogle af de ting vi nu har lært kunne være en funktion som givet to ord (strenge), printer alle bogstaver der forekommer i begge ord.

```
def in_both(word1, word2):
  #code?
```

Strenge - en sekvens af tegn

Et eksempel på brug af nogle af de ting vi nu har lært kunne være en funktion som givet to ord (strenge), printer alle bogstaver der forekommer i begge ord.

```
def in_both(word1, word2):
    for letter in word1:
        if ( letter in word2):
            print( letter )

in_both("APPLE", "PEN")
```

Strenge - en sekvens af tegn

Et eksempel på brug af nogle af de ting vi nu har lært kunne være en funktion som givet to ord (strenge), printer alle bogstaver der forekommer i begge ord.

```
def in_both(word1, word2):
    for letter in word1:
        if ( letter in word2):
            print( letter )

in_both("APPLE", "PEN")
```

```
P
P
E
```

Dictionaries

Dictionaries - Gem data og find det igen

Navnet dictionary kommer tydeligvis fra det engelske ord for ordbog, og grunden til det er at man bruger et ord til at slå op med (key) for at finde en bestemt værdi (value).

Dictionaries - Gem data og find det igen

Navnet dictionary kommer tydeligvis fra det engelske ord for ordbog, og grunden til det er at man bruger et ord til at slå op med (key) for at finde en bestemt værdi (value).

I de lister vi har set på bruges et bestemt tal (indeks) til at finde en bestemt værdi (value), dvs. værdierne har en bestemt rækkefølge.

Dictionaries - Gem data og find det igen

Navnet dictionary kommer tydeligvis fra det engelske ord for ordbog, og grunden til det er at man bruger et ord til at slå op med (key) for at finde en bestemt værdi (value).

I de lister vi har set på bruges et bestemt tal (indeks) til at finde en bestemt værdi (value), dvs. værdierne har en bestemt rækkefølge.

Dictionaries minder meget om, men vi ved ikke noget om rækkefølgen på samme måde, tilgengæld slipper vi for at holde styr på, hvilken placering en værdi har.

Dictionaries - Gem data og find det igen

Navnet dictionary kommer tydeligvis fra det engelske ord for ordbog, og grunden til det er at man bruger et ord til at slå op med (key) for at finde en bestemt værdi (value).

I de lister vi har set på bruges et bestemt tal (indeks) til at finde en bestemt værdi (value), dvs. værdierne har en bestemt rækkefølge.

Dictionaries minder meget om, men vi ved ikke noget om rækkefølgen på samme måde, tilgengæld slipper vi for at holde styr på, hvilken placering en værdi har.

I Python er "{}"og ":"vigtige for at lave et dictionary.

Her er et eksempel på en dansk–engelsk ordbog over nogle dyr.

```
translate = {"hund" : "dog", "kat" : "cat", "hest" : "horse"}
```

Dictionaries - Gem data og find det igen

Nu kan vi bruge vores dictionary til at slå oversættelserne op. Hvis man har et par "x": "y", så bruger man den forreste streng, "x", som index for at finde "y".

```
translate = {"hund" : "dog",
             "kat"  : "cat",
             "hest" : "horse"}
print( translate ["hest"])
```

Dictionaries - Gem data og find det igen

Nu kan vi bruge vores dictionary til at slå oversættelserne op. Hvis man har et par "x": "y", så bruger man den forreste streng, "x", som index for at finde "y".

```
translate = {"hund" : "dog",
             "kat"  : "cat",
             "hest" : "horse"}
print( translate ["hest"])
```

```
horse
```

Dictionaries - Gem data og find det igen

Vi kan nemt tilføje nye dyr til vores ordbog.

```
translate = {"hund" : "dog",
             "kat" : "cat",
             "hest" : "horse"}
translate["ged"] = "goat"
translate["gris"] = "pig"
print(translate["ged"])
```

Dictionaries - Gem data og find det igen

Vi kan nemt tilføje nye dyr til vores ordbog.

```
translate = {"hund" : "dog",
             "kat" : "cat",
             "hest" : "horse"}
translate["ged"] = "goat"
translate["gris"] = "pig"
print(translate["ged"])
```

```
goat
```

Dictionaries - Gem data og find det igen

Hvad sker der hvis vi vil oversætte noget vi ikke har i vores ordbog?

```
translate = {"hund" : "dog",
             "kat"  : "cat",
             "hest" : "horse"}
print( translate ["hamster"])
```

Dictionaries - Gem data og find det igen

Hvad sker der hvis vi vil oversætte noget vi ikke har i vores ordbog?

```
translate = {"hund" : "dog",
             "kat" : "cat",
             "hest" : "horse"}
print( translate ["hamster"])
```

```
KeyError

    Traceback (most
    recent call last)
<ipython-input
    -20-55577e0fe5a3>
    in <module>()
-----> 1 print(translate
    ["hamster"])

KeyError: 'hamster'
```

Dictionaries - Gem data og find det igen

Heldigvis kan vi tjekke om en værdi findes i vores dictionary.

```
translate = {"hund" : "dog",
             "kat" : "cat",
             "hest" : "horse"}
def safeTranslate(animal):
    if (animal in translate):
        print( translate[animal])
    else:
        print("<" + animal +
        "> not found")

safeTranslate("hamster")
safeTranslate("hund")
```

Dictionaries - Gem data og find det igen

Heldigvis kan vi tjekke om en værdi findes i vores dictionary.

```
translate = {"hund" : "dog",
             "kat" : "cat",
             "hest" : "horse"}
def safeTranslate(animal):
    if (animal in translate):
        print( translate[animal])
    else:
        print("<" + animal +
        "> not found")

safeTranslate("hamster")
safeTranslate("hund")
```

```
<hamster> not found
dog
```

Dictionaries - Gem data og find det igen

Hvis vi skal løbe igennem alle ting i listen kan vi gøre det på flere forskellige måder.

```
translate = {"hund" : "dog",
             "kat" : "cat",
             "hest" : "horse"}
for navn in translate:
  print("dansk: " +
        navn +
        ", engelsk: " +
        translate[navn])
```

Dictionaries - Gem data og find det igen

Hvis vi skal løbe igennem alle ting i listen kan vi gøre det på flere forskellige måder.

```
translate = {"hund" : "dog",
             "kat" : "cat",
             "hest" : "horse"}
for navn in translate :
  print("dansk: " +
        navn +
        ", engelsk: " +
        translate [navn])
```

```
dansk: hund, engelsk: dog
dansk: kat, engelsk: cat
dansk: hest, engelsk: horse
```

Dictionaries - Gem data og find det igen

I stedet for at skulle "slå op"i vores ordbog i hvert gennemløb af loopet, kan vi få key - value parret direkte ved at bruge `.items()` på dictionariet.

```
translate = {"hund" : "dog",
             "kat" : "cat",
             "hest" : "horse"}
print( translate .items())
```

Dictionaries - Gem data og find det igen

I stedet for at skulle "slå op"i vores ordbog i hvert gennemløb af loopet, kan vi få key - value parret direkte ved at bruge `.items()` på dictionariet.

```
translate = {"hund" : "dog",
             "kat" : "cat",
             "hest" : "horse"}
print( translate .items())
```

```
dict_items ([
    ('hund', 'dog'),
    ('kat', 'cat'),
    ('hest', 'horse')])
```

Dictionaries - Gem data og find det igen

I stedet for at skulle "slå op" i vores ordbog i hvert gennemløb af loopet, kan vi få key - value parret direkte ved at bruge `.items()` på dictionariet.

```
translate = {"hund" : "dog",
             "kat" : "cat",
             "hest" : "horse"}
for key, val in translate.items():
    print("dansk: " +
          key +
          ", engelsk: " +
          val)
```

Dictionaries - Gem data og find det igen

I stedet for at skulle "slå op" i vores ordbog i hvert gennemløb af loopet, kan vi få key - value parret direkte ved at bruge `.items()` på dictionariet.

```
translate = {"hund" : "dog",
             "kat" : "cat",
             "hest" : "horse"}
for key, val in translate.items():
    print("dansk: " +
          key +
          ", engelsk: " +
          val)
```

```
dansk: hund, engelsk: dog
dansk: kat, engelsk: cat
dansk: hest, engelsk: horse
```

Dictionaries - Gem data og find det igen

Dictionaries kan bruges til meget mere end som ordbog, og tingene behøver ikke være strenge, det kan være hvad som helst. Her er et eksempel på en repræsentation af en person.

```
person = {"name" : "Bamse",
          "age": 24,
          "height": 187}

if (person["height"] >= 182):
    print("You are tall "
          + person["name"])
else:
    print("You can still grow "
          + person["name"])
```

Dictionaries - Gem data og find det igen

Dictionaries kan bruges til meget mere end som ordbog, og tingene behøver ikke være strenge, det kan være hvad som helst. Her er et eksempel på en repræsentation af en person.

```
person = {"name" : "Bamse",
          "age": 24,
          "height": 187}

if (person["height"] >= 182):
    print("You are tall "
          + person["name"])
else:
    print("You can still grow "
          + person["name"])
```

```
You are tall
     Bamse
```

Dictionaries - Gem data og find det igen

Et andet eksempel kan være et dictionary som holder styr på varers priser i en butik.

```
price = {"Banana" : 10,
         "Milk" : 5,
         "Bread" : 20,
         "Salt" : 2}
print(price["Banana"])
```

Dictionaries - Gem data og find det igen

Et andet eksempel kan være et dictionary som holder styr på varers priser i en butik.

```
price = {"Banana" : 10,
         "Milk" : 5,
         "Bread" : 20,
         "Salt" : 2}
print(price["Banana"])
```

```
10
```

Dictionaries - Gem data og find det igen

Hvis vi tidligere ville have lavet et program der skulle tælle bogstaver i en lang streng, hvordan ville vi så have gjort?

Dictionaries - Gem data og find det igen

Hvis vi tidligere ville have lavet et program der skulle tælle bogstaver i en lang streng, hvordan ville vi så have gjort?

Her er et forslag:

```
s = "Lorem ipsum dolor sit amet..."
counts = [0] * 26 # [0,0,0,0,...]
for c in s.lower():
    if c == "a":
        counts[0] = counts[0] + 1
    elif c == "b":
        counts[1] = counts[1] + 1
    elif ...
        .
        .
```

Dictionaries - Gem data og find det igen

Man kunne også gøre mange andre smartere ting end det, men det smarteste er nok at bruge dictionaries.

Dictionaries - Gem data og find det igen

Man kunne også gøre mange andre smartere ting end det, men det smarteste er nok at bruge dictionaries.

```
s = "fem flade flødeboller på et fladt flødebollefad"
d = {}
for c in s:
    if c not in d:
        d[c] = 0
    d[c] = d[c] + 1
print(d)
```

Strenge og Dictionaries - Sidste bemærkninger

Som altid har vi kun vist jer et lillebitte udsnit af hvilke funktioner der findes til strenge, dictionaries, osv. Så husk, Google er kun få klik væk.