Noen Python-triks

August 24, 2022

1 Rest av divisjon

Med `%` kan vi beregne resten av en divisjon.

[1]:
```
print(3 % 2)
```

```
1
```

[2]:
```
print(5 % 3)
```

```
2
```

Vi kan bruke dette til å avgjøre om et tall kan deles på to eller ikke.

[3]:
```
a = 4

if a % 2 == 0:
    print('a er et partall')
else:
    print('a er et oddetall')
```

```
a er et partall
```

2 “List comprehension”

[4]:
```
tallrekke = [1, 2, 3, 4, 5]

ny_tallrekke = [tall * 2 for tall in tallrekke]

print(ny_tallrekke)
```

```
[2, 4, 6, 8, 10]
```

Dette tilsvarer koden:

[5]:
```
tallrekke = [1, 2, 3, 4, 5]

ny_tallrekke = []
for tall in tallrekke:
    ny_tallrekke.append(tall * 2)

print(ny_tallrekke)
```

```
[2, 4, 6, 8, 10]
```

3 Metoder som returnerer flere variabler

Først definerer vi en veldig enkel metode som returnerer to verdier. Det er også mulig å returnere flere variabler slik.

[6]:
```
def hent_tall():
    tall_a = 3
    tall_b = 7
    return tall_a, tall_b
```

Vi kan lagre de returnerte verdiene i hver sin variabel:

[7]:
```
a, b = hent_tall()
print('Tall A:', a)
print('Tall B:', b)
```

```
Tall A: 3
Tall B: 7
```

Vi kan lagre de returnerte verdiene i ett tuppel:

[8]:
```
begge_tallene = hent_tall()
print('Begge:', begge_tallene)
```

```
Begge: (3, 7)
```

4 Formaterte strenger

Når vi skal legge sammen variabler med `+` må vi sørge for at variablene er "kompatible" med hverandre. Det innebærer ofte at de er av samme type. Vi kan "legge sammen" to strenger på denne måten …

[9]:
```
resultat = '42'
print('Resultatet er ' + resultat)
```

```
Resultatet er 42
```

… men vi kan ikke gjøre det samme med en streng og et tall. Det gir en feilmelding.

[ ]:
```
resultat = 42
print('Resultatet er ' + resultat)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/var/folders/l6/bswkk8t15td75835k2hc39hw0000gn/T/ipykernel_3461/239269879.py in <module>
      1 resultat = 42
----> 2 print('Resultatet er ' + resultat)

TypeError: can only concatenate str (not "int") to str
```

Vi kan enten konvertere tallet til en streng først …

[10]:
```
resultat = 42
print('Resultatet er ' + str(resultat))
```

```
Resultatet er 42
```

… eller vi kan bruke en formatert streng, også kalt f-streng. Det gjør vi ved å skrive `f` foran strengen, og variablene i krøllparanteser `{ }`. Verdiene konverteres da automatisk til strenger.

[11]:
```
resultat = 42
print(f'Resultatet er {resultat}')
```

```
Resultatet er 42
```

Vi kan inkludere flere variabler i strengen.

[12]:
```
velkomst = 'Hei!'
resultat = 42
print(f'{velkomst} Resultatet er {resultat}.')
```

```
Hei! Resultatet er 42.
```

5 Nøkkelordsargumenter

Metoden under har har to argumenter. `a` er et posisjonsargument og `b` er et nøkkelordsargument.

[13]:
```
def multipliser(a, b=2):
    return a * b
```

Hvis vi ikke gir `b` en ny verdi i metodekallet, brukes standardverdien `2`.

[14]:
```
print(multipliser(5))
```

```
10
```

Hvis vi gir `b` en ny verdi, brukes heller den nye verdien.

[15]:
```
print(multipliser(5, b=4))
```

```
20
```

Nøkkelordsargumenter finner vi blant annet når vi leser filer: `open('tekst.txt', encoding='utf-8')`

6 Metoden `enumerate()`

[16]:
```
tokenliste = ['Python', 'er', 'gøy', '!']
```

`enumerate()` returnerer et enumerate-objekt.

[17]:
```
nummererte_token = enumerate(tokenliste)
print(nummererte_token)
```

```
<enumerate object at 0x1055c00c0>
```

Dette objektet kan det være litt vanskelig å forholde seg til. Hvis vi konverterer det til en liste, ser vi at det egentlig bare er en liste som består av tupler med tall og verdiene fra tokenlisten.

[18]:
```
print(list(nummererte_token))
```

```
[(0, 'Python'), (1, 'er'), (2, 'gøy'), (3, '!')]
```

Med `enumerate()` får vi enkel tilgang på indeksen til hvert element. Under er et eksempel hvor vi bruker indeksen til hver bokstav i alfabetstrengen til å skrive ut annenhver bokstav.

[19]:
```
alfabetet = 'abcdefghijklmnopqrstuvwxyzæøå'
for i, bokstav in enumerate(alfabetet):
    if i % 2 == 0:
        print(f'{bokstav} ({i})')
```

```
a (0)
c (2)
e (4)
g (6)
i (8)
k (10)
m (12)
o (14)
q (16)
s (18)
u (20)
w (22)
y (24)
æ (26)
å (28)
```

Vi ser at `enumerate()` teller fra null. I de aller flereste tilfeller er dette ønskelig med tanke på at Python bruker nullindeksering ellers, altså at vi teller fra null når vi indekserer. Vi kan allikevel endre starttallet med nøkkelordsargumentet `start`, dersom det viser seg nødvendig:

[20]:
```
nummererte_token = enumerate(tokenliste, start=1)
print(list(nummererte_token))
```

```
[(1, 'Python'), (2, 'er'), (3, 'gøy'), (4, '!')]
```

7 Metoden `zip()`

Med `zip()` kan vi slå sammen to lister. Listene vi slår sammen, må ha lik lengde.

[22]:
```
l = ['h', 'i', 'j']
n = ['a', 'b', 'c']

sammenslått_liste = zip(l, n)

print(list(sammenslått_liste))
```

```
[('h', 'a'), ('i', 'b'), ('j', 'c')]
```

Struktureren er den samme som når vi bruker `enumerate()`, en liste med tupler. Vi kan også iterere over denne listen.

[24]:
```
for a, b in zip(l, n):
    print(a, b)
```

```
h a
i b
j c
```

Vi kan se på et mer praktisk eksempel. La oss si at vi har en liste med bynavn og en annen liste med antall innbyggere i disse byene. Vi kan bruke `zip()` til å samle de to listene.

[26]:
```
bynavn = ['Oslo', 'København', 'Stockholm']
antall_innbyggere = [634000, 602000, 976000]

for by, antall in zip(bynavn, antall_innbyggere):
    print(f'{by} har {antall} innbyggere.')
```

```
Oslo har 634000 innbyggere.
København har 602000 innbyggere.
Stockholm har 976000 innbyggere.
```