Python Lyn Kursus

Fysisk Biokemi & Dataanalyse

Mads-Peter V. Christiansen

2025-11-06

Syntax

Python kode er læst oppe fra og ned

```
1  A = 1                                                    ①
2  B = "Denne linje er læst efter den forrige"              ②
```

① Først sættes variablen `A` lig med værdien `1`.

② Derefter defineres `B` som en `str` med en noget tekst.

Syntax

Den her kode virker

```
A = 1
B = 2
C = A + B
print(f"{C = }")
```

```
C = 3
```

Da vi først definere `A = 1` og `B = 2` og derefter bruger dem.

Syntax

Så hvad sker der hvis vi prøver noget hvor rækkefølgen er forkert?

```
1 A = 1
2 C = A + B
3 B = 2
4 print(f"{C = }")
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[206], line 2
      1 A = 1
----> 2 C = A + B
      3 B = 2
      4 print(f"{C = }")

NameError: name 'B' is not defined
```

Det giver en fejl! Dette er selvfølgelig frusterende, men det kan vendes om til at være meget hjælpsomt.

Alle får disse fejl - uanset erfaring - jeg får **mange** fejl!

Ikke noget galt i at få fejl - brug det til at finde ud af hvad der går galt.

Data typer & Variable

Python har forskellige **typer** af data

```
1 et_hel_tal = 1
2 et_kommatal = 3.2
3 en_streng = "Dette er en streng, den indeholder tekst"
```

Forskellige typer kan forskellige ting

```
1 et_hel_tal + et_kommatal
```

```
4.2
```

Python forstår at vi kan lægge tal sammen, selvom de er forskellige typer.

```
1 et_hel_tal + en_streng
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[209], line 1
----> 1 et_hel_tal + en_streng

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Fejl igen. Overvej hvad fejlen betyder!

Flere typer: `DataFrame`

En `DataFrame` er en type der kan tænkes på som en tabel eller et excel ark. Her er en `DataFrame` med to kolonner og 5 rækker.

```
1 print(df)
```

```
   Kolonne 1  Kolonne 2
0          1          2
1          2          4
..       ...        ...
3          4         16
4          5         25

[5 rows x 2 columns]
```

Vi kan spørge efter en af kolonnerne ved at **indeksere** med dens navn.

```
1 col2 = df['Kolonne 2']
2 print(col2)
```

```
0     2
1     4
     ..
3    16
4    25
Name: Kolonne 2, Length: 5, dtype: int64
```

Flere typer: `DataFrame`

Vi kan lave forskellige operationer med kolonner

```python
col1 = df['Kolonne 1']
col2 = df['Kolonne 2']

sum_of_cols = col1 + col2
print(sum_of_cols)
```

```
0     3
1     6
     ..
3    20
4    30
Length: 5, dtype: int64
```

$$\begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{pmatrix} + \begin{pmatrix} 2 \\ 4 \\ 9 \\ 16 \\ 25 \end{pmatrix} = \begin{pmatrix} 1+2 \\ 2+4 \\ 3+9 \\ 4+16 \\ 5+25 \end{pmatrix} = \begin{pmatrix} 3 \\ 6 \\ 12 \\ 20 \\ 30 \end{pmatrix}$$

Dette er en “elementwise”-operation.

Flere typer: `DataFrame`

Kan også gange dem sammen.

```
col1 = df['Kolonne 1']
col2 = df['Kolonne 2']

prod_of_cols = col1 * col2
print(sum_of_cols)
```

```
0     3
1     6
     ..
3    20
4    30
Length: 5, dtype: int64
```

$$\begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{pmatrix} + \begin{pmatrix} 2 \\ 4 \\ 9 \\ 16 \\ 25 \end{pmatrix} = \begin{pmatrix} 1 \cdot 2 \\ 2 \cdot 4 \\ 3 \cdot 9 \\ 4 \cdot 16 \\ 5 \cdot 25 \end{pmatrix} = \begin{pmatrix} 2 \\ 8 \\ 27 \\ 64 \\ 125 \end{pmatrix}$$

Parenteser

Python bruger parenteser til forskellige ting.

1. Runde parenteser `(...)` bruges i udregniner som almindeligt.
2. Firkant parenteser `[...]` bruges til **indeksering**
3. Runde parenteser bruges også til funktions-kald `print('Noget tekst')`

Parenteser: Beregning

Hvis vi vil regne dette udtryk

$$\frac{A + B + C}{2 \cdot C}$$

Skal vi bruge de runde `(...)` parenteser.

```
1 (A + B + C) / (2*C)
```

```
42.0
```

Hvis vi gør det forkert - bliver svaret ikke rigtigt!

```
1 A + B + C / (2*C)
```

```
83.5
```

Parenteser: Beregning - Fortsat

Altid en god ide at teste med simple værdier

$$\frac{A + B + C}{2 \cdot C} = \frac{1 + 1 + 1}{2 \cdot 1} = \frac{3}{2} = 1.5$$

```
1 A = 1
2 B = 1
3 C = 1
4 test_1 = (A + B + C) / (2*C)
5 test_2 = A + B + C / (2*C)
6
7 print(test_1)
8 print(test_2)
```

```
1.5
2.5
```

Parenteser: Indeksering

En anden type brug af paranteser er **indeksering**.

Indeksering handler om at trække det pågældende element ud af et "data-objekt".

Tænk: Slå op ved indeks 200 Her er "data-objektet" kartotekket.

Parenteser: Indeksering

En anden type brug af paranteser er **indeksering**.

Vi kan indeksere (eller "slå op i") en `DataFrame` - f.eks. hvis vi gerne vil have `Kolonne 2`.

```
1 col2 = df['Kolonne 2']
2 print(col2)
```

```
0     2
1     4
     ..
3    16
4    25
Name: Kolonne 2, Length: 5, dtype: int64
```

Nu er "data-objektet" en `DataFrame`

```
1 print(df)
```

```
   Kolonne 1  Kolonne 2  Kolonne 3
0          1          2          3
1          2          4          6
..       ...        ...        ...
3          4         16         20
4          5         25         30

[5 rows x 3 columns]
```

Parenteser: Indeksering

En anden type brug af paranteser er **indeksering**.

Vi kan også indeksere yderligere og tage et specifikt element.

```
col2 = df['Kolonne 2']
element_3 = col2[2]
print('Element 3 er', element_3)
```

```
Element 3 er 9
```

Nu er "data-objektet" en `DataFrame`

```
print(df)
```

```
    Kolonne 1  Kolonne 2  Kolonne 3
0           1          2          3
1           2          4          6
..        ...        ...        ...
3           4         16         20
4           5         25         30

[5 rows x 3 columns]
```

Parenteser: Funktioner

Parenteser bruges ydermere til at give inputs/argumenter til funktioner

Jeg har et par gange f.eks. brugt funktionen `print`.

```
1 print('Dette er det første argument')
```

```
Dette er det første argument
```

Forskellige funktioenr tager forskellige argumenter

```
1 print('Dette er det første argument', 'Dette er det andet argument')
```

```
Dette er det første argument Dette er det andet argument
```

Print funktionen sætter alle givne argumenter sammen og printer en samlet tekst.

Parenteser: Funktioner

Vi kan lave en funktion der tager et argument

```
1 def min_funktion(a):
2     return 2 * a
```

Lad os forsøge at bruge den

```
1 min_funktion # Uden parenteser for vi bare at vide hvor Python har gemt funktionen.
```

```
<function __main__.min_funktion(a)>
```

Hvis vi giver **et** argument går det godt!

```
1 min_funktion(1)
```

```
2
```

Hvis vi giver to argumenter går det galt

```
1 min_funktion(1, 2)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[230], line 1
----> 1 min_funktion(1, 2)
```

Parenteser: Funktioner

Lad os prøve det samme med en funktion der tager 2 argumenter

```
1 def min_anden_funktion(a, b):
2     return a + b
```

Funktionen forventer to argumenter og alt går godt hvis vi giver to

```
1 min_anden_funktion(1, 2)
```

```
3
```

Hvis vi kun giver denne funktion et argument får vi en fejl

```
1 min_anden_funktion(1)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[233], line 1
----> 1 min_anden_funktion(1)

TypeError: min_anden_funktion() missing 1 required positional argument: 'b'
```

Opsummering om paretenser

Matematik

```
1  (1 + 1 + 1) / (3 + 4)
```

Brug runde parenteser `(...)` til matematik. God ide at checke med simple tal!

Indeksering

```
1  col1 = df['Kolonne 1']
2  element_3 = col1[3]
```

Brug firkant parenteser `[...]` til indeksering.

Funktioner

```
1  en_funktion(argument_1, argument_2)
```

Funktioner kaldes med runde parenteser. Argumenter er kommaseparerede

In practice: Step 1

Fra TØ opgaven: Proteins in blood plasma

Kig på datasættet og tænk over hvad det indeholder!

```
1 df
```

	A280_protein1_healthy	A280_protein1_patient	A280_protei
0	0.086241	0.014437	0.035768
1	0.095333	0.014813	0.040662
...	...	...	...
498	0.096186	0.014589	0.040299
499	0.079806	0.013657	0.030360

500 rows × 4 columns

In practice: Step 2

Fra TØ opgaven: Proteins in blood plasma

Tænk over hvad opgaven spørg efter, og hvad der er oplyst.

> ## (b) Calculate concentrations
>
> Calculate the molar concentration of the two proteins in all samples, the light path for every measurement is 0.1 cm.
>
> Always a good idea to assign known values to variables

```
protein_1_ext_coeff = 180000
protein_2_ext_coeff = 80000
l = 0.1
```

...

```
df['protein1_healthy_molar_conc'] = ... # Calculate for concentration in healthy f
... # Your code that updates the data frame with the 3 other new columns.
display(df)
```

In practice: Step 2

Fra TØ opgaven: Proteins in blood plasma

Tænk over hvad opgaven spørg efter, og hvad der er oplyst.

> ## (b) Calculate concentrations
>
> Calculate the molar **concentration** of the two proteins in all samples, the **light path** for every measurement is **0.1 cm**.
>
> Always a good idea to assign known values to variables

```
protein_1_ext_coeff = 180000 # Disse er givet ovenfor
protein_2_ext_coeff = 80000  # -//-
l = 0.1 # Det er værdien givet fra opgave teksten.
```

...

```
df['protein1_healthy_molar_conc'] = ... # Calculate for concentration in healthy f
... # Your code that updates the data frame with the 3 other new columns.
display(df)
```

In practice: Step 3

Løs det biokemiske problem

> Kender absorbans A, lys vej længde l, extinctions coefficenter ϵ. Vil gerne regne molar concentrationen c. Hvilken ligningen kunne gøre det?
>
> Beer-Lamberts lov
>
> $$A = \epsilon \cdot l \cdot c$$
>
> Isoler den relevante størrelse
>
> $$A = \epsilon \cdot l \cdot c \rightarrow c = \frac{A}{\epsilon \cdot l}$$

In practice: Step 4

Nu skal vi så skrive koden for at gøre dette.

Det givne kode var

```
df['protein1_healthy_molar_conc'] = ... # Calculate for concentration in healthy for p
... # Your code that updates the data frame with the 3 other new columns.
df
```

Tænk over hvad der svarer til hvad imellem ligning og kode

$A \longrightarrow$ `df['A280_protein1_healthy']`

$\epsilon \longrightarrow$ `protein_1_ext_coeff`

$l \longrightarrow$ `l`

Så

$$c = \frac{A}{\epsilon \cdot l}$$

Bliver til

```
df['protein1_healthy_molar_conc'] = df['A280_protein1_healthy'] / (protein_1_ext_coeff
```

In practice: Step 5

Tænk over om resultatet giver mening

Vi har regnet

```
print(df['protein1_healthy_molar_conc'][0]) # Her kigger jeg på den første værdi.
```

```
4.791141975308642e-06
```

Med

```
print(df['A280_protein1_healthy'][0])
print(f"{protein_1_ext_coeff = }")
print(f"{l = }")
```

```
0.08624055555555556
protein_1_ext_coeff = 180000
l = 0.1
```

Mentalitet

Tænk på det som at i er en mekaniker lærling der er givet en motor.

Med den vigtige forskelle er at i kan **ikke** ødelægge noget.

Mentalitet

Opgaven fra før, men med en fejl

```
  1 df('protein1_healthy_molar_conc') = df['A280_protein1_healthy'] / (protein_1_ext_coeff
```

```
  Cell In[241], line 1
    df('protein1_healthy_molar_conc') = df['A280_protein1_healthy'] / (protein_1_ext_coeff * l)
    ^
SyntaxError: cannot assign to function call here. Maybe you meant '==' instead of '='?
```

Okay, det giver en fejl - prøver at gøre som fejlen siger

```
  1 df('protein1_healthy_molar_conc') == df['A280_protein1_healthy'] / (protein_1_ext_coef
```

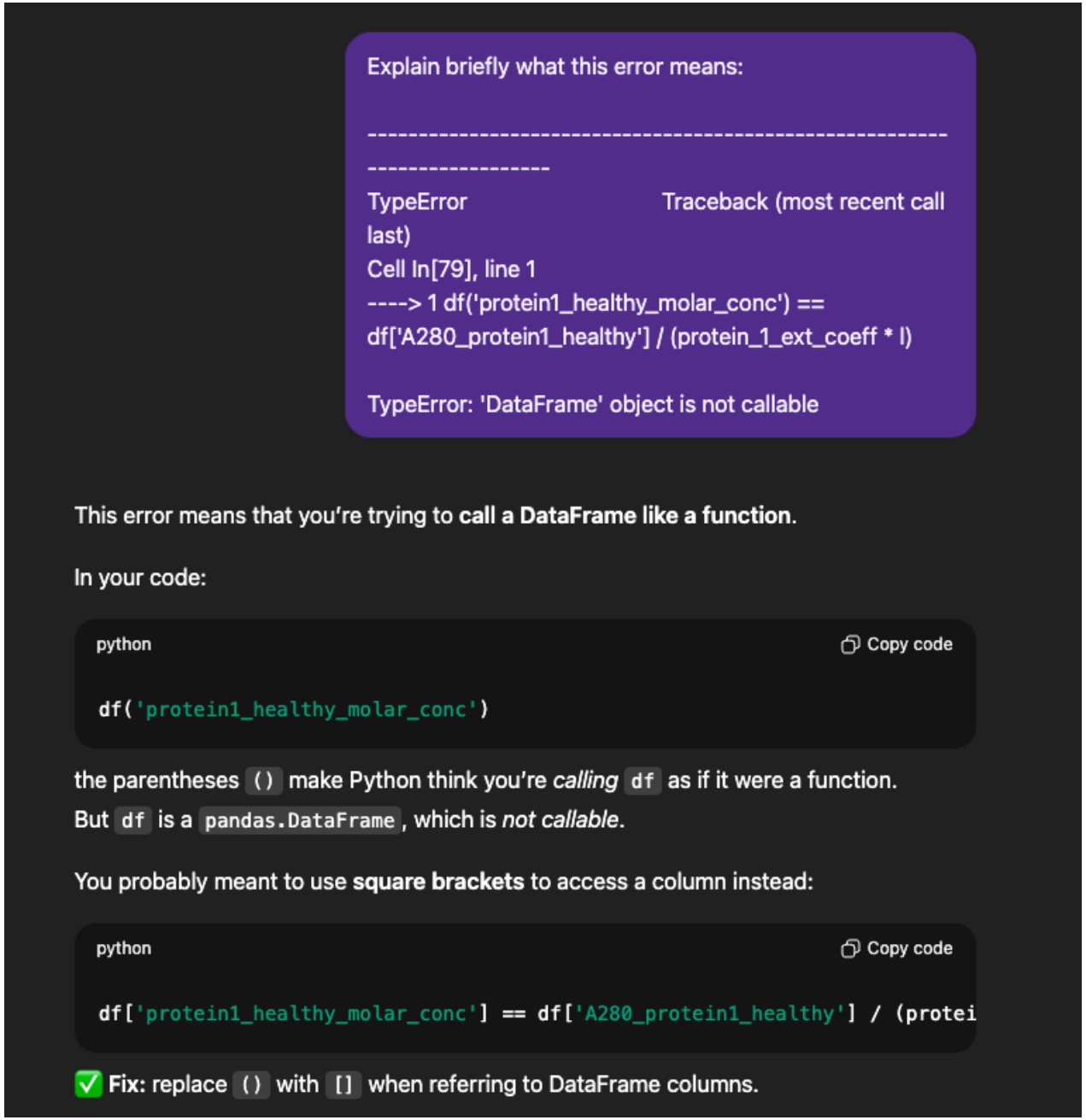
```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[242], line 1
----> 1 df(                                ) == df['A280_protein1_healthy'] / (protein_1_ext_coeff

TypeError: 'DataFrame' object is not callable
```

Så, det giver en anden fejl. Den fejl er lidt sværere at forstå hvad

Mentalitet

Før havde det nu været tid til at Google' og forhåbenligt få et svar fra stackoverflow. Men det er nemmere nu om dage.


Explain briefly what this error means:

TypeError Traceback (most recent call last)
Cell In[79], line 1
----> 1 df('protein1_healthy_molar_conc') == df['A280_protein1_healthy'] / (protein_1_ext_coeff * l)
TypeError: 'DataFrame' object is not callable
This error means that you're trying to call a DataFrame like a function.
In your code:
python
Copy code
df('protein1_healthy_molar_conc')
the parentheses () make Python think you're calling df as if it were a function.
But df is a pandas.DataFrame, which is not callable.
You probably meant to use square brackets to access a column instead:
python
Copy code
df['protein1_healthy_molar_conc'] == df['A280_protein1_healthy'] / (protei
✅ Fix: replace () with [] when referring to DataFrame columns.

Mentalitet

Okay, skal bruge firkant parenteser

```
1 df['protein1_healthy_molar_conc'] = df['A280_protein1_healthy'] / (protein_1_ext_coeff
2 print(df['protein1_healthy_molar_conc'])
```

```
0      0.000005
1      0.000005
         ...
498    0.000005
499    0.000004
Name: protein1_healthy_molar_conc, Length: 500, dtype: float64
```

Helpdesk

Kom til `Python Helpdesk`

Tirsdage: `10.00-12.00`

Lokation: `Kælderen`

Person: Mig (`Mads-Peter`)

...

...

...

...

9/10 tandlæger siger jeg er flink og 10/10 siger jeg har dårlig humor.