# Lær Python dag 3 - modul 1

## Institut for Matematik og Datalogi, Syddansk Universitet

Mikkel Juul Vestergaard

https://moggel12.github.io/PythonWorkshop2021/

23. maj, 2021

# Indhold



# Hvad lærte I sidst?

Brug tre minutter på at snakke om hvad i har lært op til nu, i breakout rooms.

- Hvad er en løkke?
- Hvad er en liste?
- Hvad kan vi med strenge?
- Andet?

# Objekter og metoder (kort)

# Objekter og metoder

I har prøvet at bruge funktioner som:

```
x = [1, 2]
x.append(3)
print(x)
```

# Objekter og metoder

I har prøvet at bruge funktioner som:

```
x = [1, 2]
x.append(3)
print(x)
```

Output:

```
[1, 2, 3]
```

# Objekter og metoder

I har prøvet at bruge funktioner som:

```
x = [1, 2]
x.append(3)
print(x)
```

Output:

```
[1, 2, 3]
```

Har I undret jer over hvad punktummet betyder?

# Objekter og metoder

I har prøvet at bruge funktioner som:

```
x = [1, 2]
x.append(3)
print(x)
```

Output:

```
[1, 2, 3]
```

Har I undret jer over hvad punktummet betyder? Hvorfor skriver man ikke:

```
append(x, 3)
```

# Objekter og metoder

Hvad sker der hvis man skriver:

```
x = 2
x.append(3)
```

# Objekter og metoder

Hvad sker der hvis man skriver:

```
x = 2
x.append(3)
```

Output:

```
...

AttributeError: 'int' object has no attribute 'append'
```

# Objekter og metoder

Hvad sker der hvis man skriver:

```
x = 2
x.append(3)
```

Output:

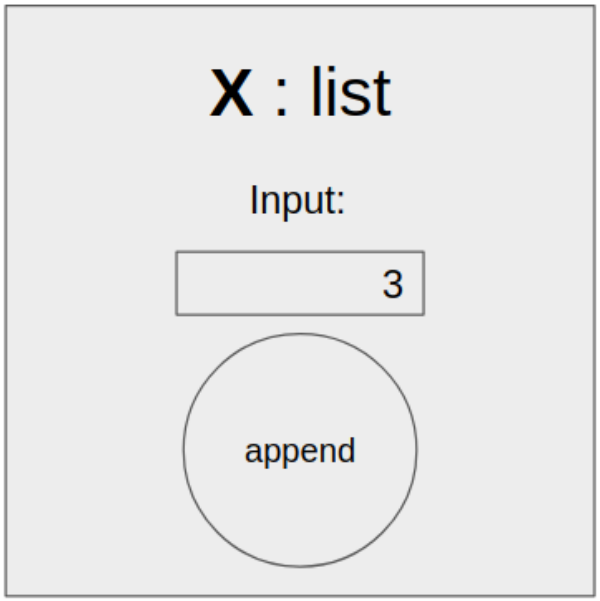
```
...

AttributeError: 'int' object has no attribute 'append'
```

Men så må det være tilfældet at - `'list' object has attribute 'append'`.
Hvad end det så betyder.

# Objekter og metoder

Man kan tænke på det som et "interface" på "kassen".

```
x = [1,2]
x.append(3)
```

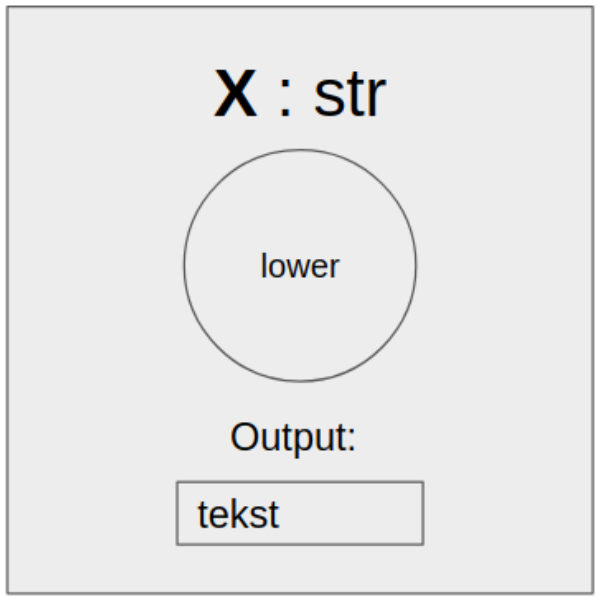



Eksempler på flere metoder kan findes på:
`https://docs.python.org/3/tutorial/datastructures.html`

# Objekter og metoder

Et andet eksempel kunne være

```
x = "TEKST"
x.lower()
```




Eksempler på flere metoder kan findes på:
`https://docs.python.org/3/library/stdtypes.html#str`

# Dictionaries

# Dictionaries - Gem data og find det igen

Navnet dictionary kommer fra det engelske ord for ordbog. Her slåes et ord op med et *nøgleord* (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 lister, men de har ikke en rækkefølge, tilgengæld slipper vi for at holde styr på, hvilken placering en værdi har.
I python er "{}" og ":" vigtige for at lave en dictionary.

# Dictionaries - Gem data og find det igen

Navnet dictionary kommer fra det engelske ord for ordbog. Her slåes et ord op med et *nøgleord* (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 lister, men de har ikke en rækkefølge, tilgengæld slipper vi for at holde styr på, hvilken placering en værdi har.
I python er "{}" og ":" vigtige for at lave en dictionary.
Her er et eksempel på en dansk–engelsk ordbog over nogle dyr.

```
dyr = {"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".

```
dyr = {"hund" : "dog",
       "kat" : "cat",
       "hest" : "horse"}
print(dyr["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".

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

```
horse
```

# Dictionaries - Gem data og find det igen

Vi kan nemt tilføje nye dyr til vores ordbog.

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

# Dictionaries - Gem data og find det igen

Vi kan nemt tilføje nye dyr til vores ordbog.

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

```
goat
```

# Dictionaries - Gem data og find det igen

Vi kan også tjekke om en værdi findes i vores dictionary.

```
dyr = {"hund" : "dog",
       "kat" : "cat",
       "hest" : "horse",
       "ged" : "goat",
       "gris" : "pig"}
print("hamster" in dyr)
print("hund" in dyr)
```

# Dictionaries - Gem data og find det igen

Vi kan også tjekke om en værdi findes i vores dictionary.

```
dyr = {"hund" : "dog",
       "kat" : "cat",
       "hest" : "horse",
       "ged" : "goat",
       "gris" : "pig"}
print("hamster" in dyr)
print("hund" in dyr)
```

```
False
True
```

# Dictionaries - Gem data og find det igen

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

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

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

```
dyr = {"hund" : "dog",
       "kat" : "cat",
       "hest" : "horse"}
for key, val in dyr.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.

```
dyr = {"hund" : "dog",
       "kat" : "cat",
       "hest" : "horse"}
for key, val in dyr.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": 23,
          "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": 23,
          "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

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 = "Lorem ipsum dolor sit amet..."
d = {}
for c in s:
    if c not in d:
        d[c] = 0
    d[c] = d[c] + 1
```

# 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.

# Filer

# Filer og programmering

Filer bruges i flere sammenhænge:

- Gemme data til senere brug (persistent data)
- Skriv output fra program til en filer
- Tag input fra en fil

# Åben/luk en fil i python

Vi åbner en fil med methoden `open()` som returnerer et filobjekt:

```
f = open(<file>, <mode>)
```

Tager parametrene `file` og `mode`:

- File: Stien til den fil som skal åbnes (string).
- Mode: Hvordan skal filen bruges "r" for læs, "w" for skriv.

Når man er færdig med filen skal den lukkes (for at frigøre ressourcer):

```
f.close()
```

# Læs fra en fil

Læs filen `min_fil.txt` på følgende vis:

```
f = open("min_fil.txt", "r")
print(f.read())
f.close()
```

Dette giver output:

```
Hej med jer
Dette er en fil
```

# Læs fra en fil

Vi kan også læsen filen linje for linje:

```
f = open("min_fil.txt", "r")
line = f.readline()
while line:
    print(line.rstrip())
    line = f.readline()
f.close()
```

Kan også gøres således:

```
f = open("min_fil.txt", "r")
for line in f:
    print(line.rstrip())
f.close()
```

`rstrip()` fjerner whitespace for enden af linjen (undgår ekstra linjeskift).

# Skriv til en fil

Skriv til filen `min_fil.txt` på følgende vis:

```
f = open("min_fil.txt", "w")
f.write("Lær python!\n")
f.write("IMADA SDU")
f.close()
```

`\n` er en new-line-karakter og tilføjer derved et linjeskift i filen.
Filen ser nu således ud:

```
Lær python!
IMADA SDU
```

# Exceptions

Hvis vi skriver til en fil som ikke eksistere oprettes en ny.
Hvad sker der hvis vi prøver at læse en fil som ikke eksisterer?

# Exceptions

Hvis vi skriver til en fil som ikke eksistere oprettes en ny.
Hvad sker der hvis vi prøver at læse en fil som ikke eksisterer?
Her prøver vi at læse den ikke eksisterende fil
`ikke_eksisterende_fil.txt`:

```
File "C:/Users/SBK/Documents/GitHub/Scripts/python_course/test.py",
  line 18, in <module>
    f = open(" ikke_eksisterende_fil .txt", "r")
FileNotFoundError: [Errno 2] No such file or directory :
  ' ikke_eksisterende_fil .txt'
```

Programmet fejler! og stopper med en fejl.

# Exceptions

Vi kan dog fange denne fejl (aka. exception), og give en bedre fejlmeddelelse, eller alt efter fejlen rette op på den.
Dette gøres via `try` og `except`:

```
f = None
try:
    f = open(" ikke_eksisterende_fil .txt", "r")
    print(f.read())
    f.close()
except:
    if f is not None:
        f.close()
    print(" filen  kunne ikke åbnes") # fejlmeddelelse
```

Vi er her meget forsigtige i except-blokken og sørger for at lukke filen, hvis der er blevet initaliseret.

# Exceptions

Vi kan også fange en specifik fejl:

```
f = None
try:
    f = open(" ikke_eksisterende_fil .txt", "r")
    print(f.read())
    f.close()
except FileNotFoundError:
    if f is not None:
        f.close()
    print(" filen kunne ikke åbnes")
```

# Afsluttende bemærkning

I har nu lært om bl.a.

- Typer & variabler
- If-sætninger
- Funktioner
- Lister
- Løkker
- Strenge
- Dictionaries
- Filer

Hvilket i princippet er mere end rigeligt til at kunne gøre "alt".

# Afsluttende bemærkning

Det vigtigste vi ikke har lært om er hvordan man laver sine egne ”klasser”, sine egne typer.

# Afsluttende bemærkning

Det vigtigste vi ikke har lært om er hvordan man laver sine egne "klasser", sine egne typer.
Derudover er der masser af detaljer om de ting vi har lært som også kan udforskes.

# Afsluttende bemærkning

Det vigtigste vi ikke har lært om er hvordan man laver sine egne "klasser", sine egne typer.
Derudover er der masser af detaljer om de ting vi har lært som også kan udforskes.
Læs andres kode, læs bøger, se videoer, lav tutorials, lav et projekt og find hjælp på nettet.