Lær Python dag 2 - modul 1

Institut for Matematik og Datalogi, Syddansk Universitet

Mikkel Juul Vestergaard

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

16. maj, 2021

Indhold

Recap

Løkker

Lister

Hvad lærte I sidst?

- Hvordan opretter man en variabel?
- Hvordan kan man tjekke typen på en variabel?
- Hvordan laver man et funktionskald?
- Hvad er en funktion?
- Hvad kan man bruge funktioner til?

Løkker

Tæl til 5

Hvordan kan vi skrive kode som tæller fra 1-5 (printer tallene)?

Tæl til 5

```
print(1)
print(2)
print(3)
print(4)
print(5)
```

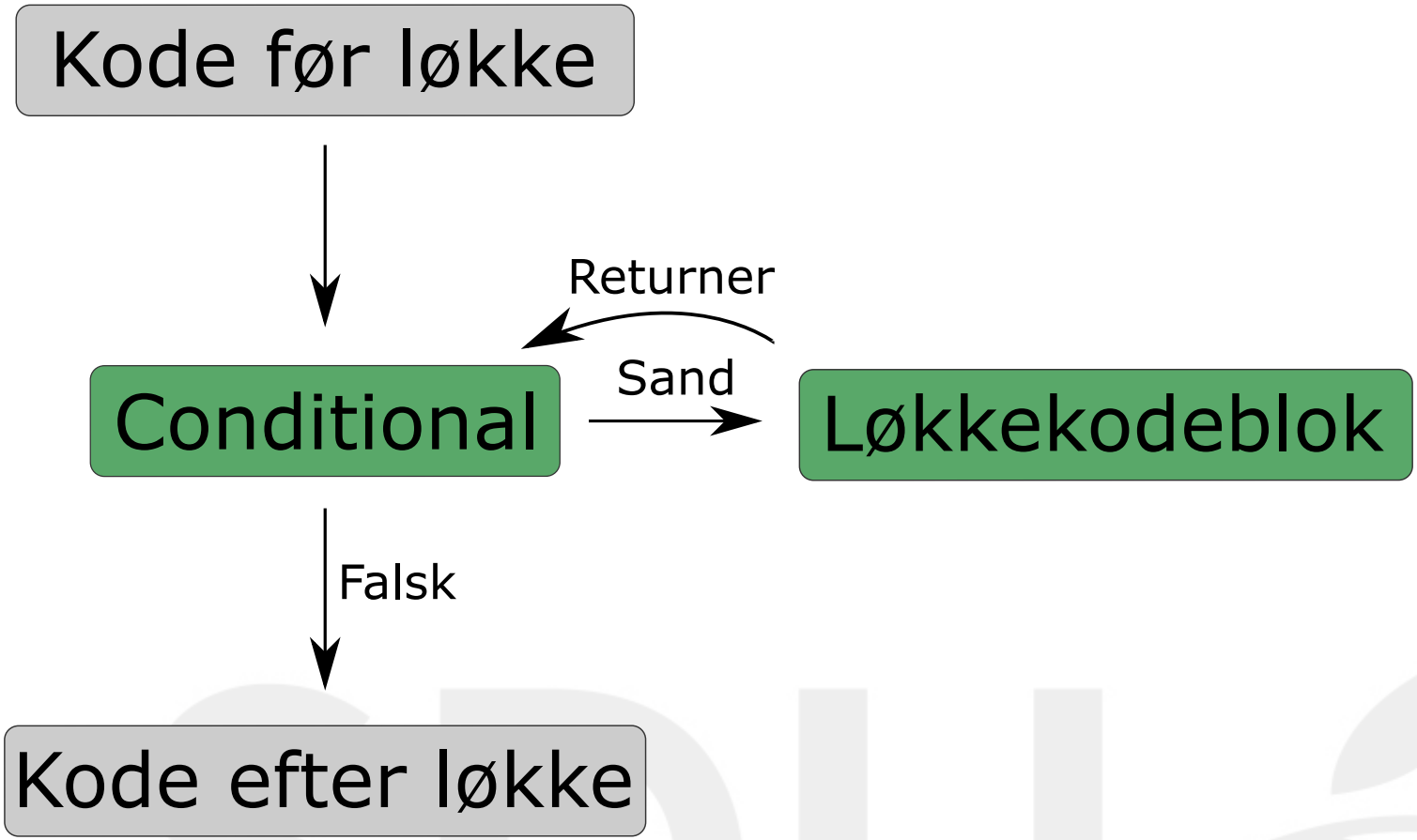
```
1
2
3
4
5
```

Ovenstående virker umiddelbart ret besværligt, især hvis det er en større mængde instruktioner vi vil have gentaget flere gange! Eller et variabelt antal gange!

Kan vi ændre dette til at være mere overskueligt?

Løkker

Vi kan gøre det med løkker!



For loop

Gentagelse ved hjælp af for-løkke:

```
for <variabel> in range(<antal>):
    <instruktion>
    <instruktion>
    <instruktion>
```

For loop

Gentagelse ved hjælp af for-løkke:

```
for <variabel> in range(<antal>):
  <instruktion>
  <instruktion>
  <instruktion>
```

Eksempel:

```
for i in range(5):
  print(i)
```

```
0
1
2
3
4
```

For loop

Men hvad hvis vi gerne vil tælle fra 1 til 5?

For loop

Men hvad hvis vi gerne vil tælle fra 1 til 5?

```
for i in range(1, 6):
  print(i)
```

```
1
2
3
4
5
```

Læg mærke til grænserne. Off-by-one er en klassisk fejl at lave...
Begynd at vænne jer til at tælle fra 0.

For loop

Vi kan også tage skridt af større end én:

```
for i in range(1, 10, 3):
  print(i)
```

```
1
4
7
```

While loop

En anden type løkke er `while`-løkken. Denne kører så længe en betingelse er overholdt:

```
while (<condition>):
    <instruktion>
    <instruktion>
    <instruktion>
```

Tilbage til vores tælleeksempel:

```
i = 1
while (i < 6):
    print(i)
    i = i + 1
```

```
1
2
3
4
5
```

While loop

Hvad gør følgende løkke?

```
i = 1
while (True):
    print(i)
    i = i + 1
```

While loop

Hvad gør følgende løkke?

```
i = 1
while (True):
    print(i)
    i = i + 1
```

```
1
2
3
4
5
6
7
8
.
.
.
```

Det kører for evigt!

While loop

Giver while-true løkker mening? Hvor kan de bruges?

While loop

Giver while-true løkker mening? Hvor kan de bruges?

- Kode som skal køre hele tiden (fx operativ system)
- Når man ikke kender antallet af gange koden skal gentages
  - Hvordan brydes løkken så?

While loop

Giver while-true løkker mening? Hvor kan de bruges?

- Kode som skal køre hele tiden (fx operativ system)
- Når man ikke kender antallet af gange koden skal gentages
  - Hvordan brydes løkken så?

Brug `break`'s:

```
i = 1
while(True):
    print(i)
    i = i + 1
    if (i > 5):
        break
```

While loop

Giver while-true løkker mening? Hvor kan de bruges?

- Kode som skal køre hele tiden (fx operativ system)
- Når man ikke kender antallet af gange koden skal gentages
  - Hvordan brydes løkken så?

Brug `break`'s:

```
i = 1
while(True):
    print(i)
    i = i + 1
    if (i > 5):
        break
```

```
1
2
3
4
5
```

While loop

Vi kan også springe videre til næste iteration med `continue`:

```
i = 1
while(True):
    if (i == 2):
        i = i + 1
        continue
    print(i)
    i = i + 1
```

```
1
3
4
5
```

Nested loops

Man kan også have løkker i løkker:

```
for i in range(5):
    for j in range(2):
        print(j)
```

Nested loops

Man kan også have løkker i løkker:

```
for i in range(5):
    for j in range(2):
        print(j)
```

```
0
1
0
1
0
1
0
1
0
1
```

Datastrukturer

”Datastrukturer er en fællesbetegnelse for data, der er organiserede i elementer, som kan tilføjes eller fjernes fra strukturen”
- Wiki

Datastrukturer

Altså en måde at gemme data på en struktureret måde.
En meget simpel datastruktur er en liste (en sekvens af data).

Eksempel på lister i python:

```
list1  = [1, 2, 3, 4]
lits2  = ["hej", "med", "jer"]
list3  = [True, False, True]
list3  = [[1,2], [" lister "], ["af", " lister "]]
list4  = [1, "hej", False]
```

Bemærk vi kan have lister med elementer af blandede typer. Og nestede lister.

Indeksering i lister

Indeksering foregår med med de kantede parenteser:

```
mylist = [4, 0, 3, 8]
print( mylist [2])
```

Indeksering i lister

Indeksering foregår med med de kantede parenteser:

```
mylist = [4, 0, 3, 8]
print(mylist[2])
```

```
3
```

Vent hvad? Vi tæller fra 0!

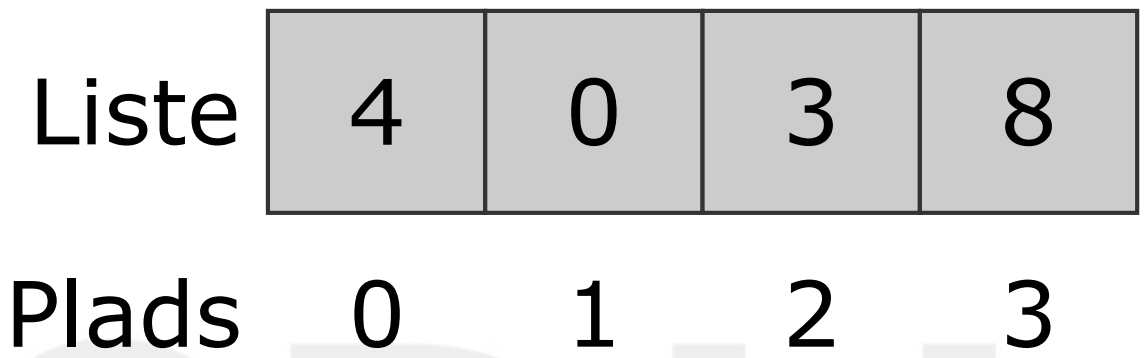
Indeksering i lister

Indeksering foregår med med de kantede parenteser:

```
mylist = [4, 0, 3, 8]
print(mylist[2])
```

```
3
```

Vent hvad? Vi tæller fra 0!



Slicing

Vi kan tage en del af en liste ved hjælp af slicing:

```
mylist = [4, 0, 3, 8]
print( mylist [1:3])
```

Slicing

Vi kan tage en del af en liste ved hjælp af slicing:

```
mylist = [4, 0, 3, 8]
print( mylist [1:3])
```

```
[0, 3]
```

Bemærk her at element 1 og 2 printes. For `[n:m]` inkluderes det `n`'te element og det `m`'te eksluderes.

Iteration af en liste

Et gennemløb af en liste er en typisk operation:

```
mylist = [1, 2, 3, 4]
for elm in mylist:
    print(elm)
```

```
1
2
3
4
```

Her er `elm` en variabel. Kan også gøres med de allerede lærte løkker som while- og forløkker.

Ændring af element

Lister er mutable. Dvs. vi kan ændre dets elementer.

```
mylist = [1, 2, 4]
print(mylist)
mylist[0] = 8
print(mylist)
```

```
[1, 2, 4]
[8, 2, 4]
```

Bemærk igen at første plads i listen er plads nummer 0.

Listemetoder

Tilføj til en liste:

```
mylist = [1, 2, 4]
print(mylist)
mylist.append(5)
print(mylist)
```

```
[1, 2, 4]
[1, 2, 4, 5]
```

Listemetoder

Tilføj til en liste:

```
mylist = [1, 2, 4]
print(mylist)
mylist.append(5)
print(mylist)
```

```
[1, 2, 4]
[1, 2, 4, 5]
```

Fjern fra en liste

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

```
[1, 2, 4]
[2, 4]
```

Listemetoder

Længde af en liste.

```
mylist = [1, 2, 4]
x = len(mylist)
print(x)
```

```
3
```

Listemetoder

Længde af en liste.

```
mylist = [1, 2, 4]
x = len(mylist)
print(x)
```

```
3
```

Concatenation:

```
mylist1 = [1, 2]
mylist2 = [3, 4]
mylist3 = mylist1 + mylist2
print(mylist3)
```

```
[1, 2, 3, 4]
```

Listemetoder

Multiplication:

```
mylist1 = [1, 2]
mylist2 = mylist1 * 3
print(mylist2)
```

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

Listemetoder

Multiplication:

```
mylist1 = [1, 2]
mylist2 = mylist1 * 3
print(mylist2)
```

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

Tjek om element i en liste:

```
mylist = [1, 2, 3, 4, 5]
print(2 in mylist)
print(7 in mylist)
```

```
True
False
```

Lidt af det hele

Et eksempel

```
yndlings = []
for i in range(3):
    x = int(input("Indtast et af dine yndlingstal \n"))
    yndlings.append(x)

if (42 in yndlings):
    print("42 er lækkert")
else:
    print("Du har ikke mit yndlingstal i din top 3 :(")
```

Range

Det kan være brugbart at tænke på `range` således:

range(5) == [0, 1, 2, 3, 4]

Range

Det kan være brugbart at tænke på `range` således:

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

Når man kigger på

```
for i in range(5):
    print(i)
```

```
0
1
2
3
4
```

Lidt af det hele

Hvis man har brug for indekset af et element gør man ofte således:

```
mylist = [4, 0, 3, 8]
for i in range(len(mylist)):
    print("På plads " + str(i) + " er der " + str(mylist[i]))
```

Lidt af det hele

Hvis man har brug for indekset af et element gør man ofte således:

```
mylist = [4, 0, 3, 8]
for i in range(len(mylist)):
    print("På plads " + str(i) + " er der " + str(mylist[i]))
```

Output:

```
På plads 0 er der 4
På plads 1 er der 0
På plads 2 er der 3
På plads 3 er der 8
```

Are you not entertained?

Flere metoder og eksempler kan findes her:
`https://docs.python.org/3/tutorial/datastructures.html`