Lær Python dag 2 - modul 1

Løkker og lister

Jonas Bamse Andersen

Institut for Matematik og Datalogi - IMADA
Syddansk Universitet

Indhold

1. Recap
2. Løkker
3. Lister

Recap

Hvad lærte I sidst?

Diskutér to minutter med sidemanden, hvad I lærte sidst.

- Hvad er en type?
- Hvad er en variabel?
- Hvad er en if-sætning?
- Hvad er en funktion?

Løkker

Tæl til 5

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

Tæl til 5

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

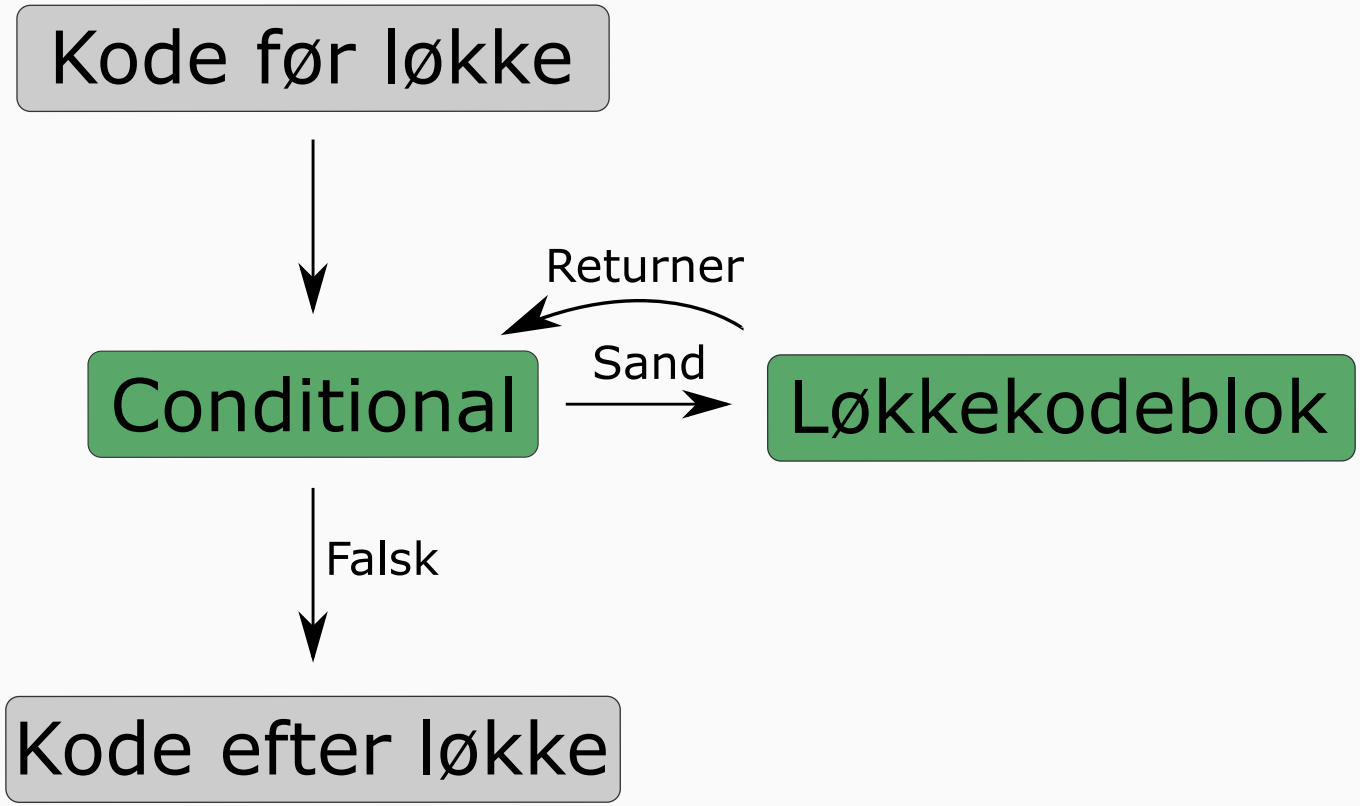
```
print(1)
print(2)
print(3)
print(4)
print(5)
```

```
1
2
3
4
5
```

Virker umiddelbart lidt besværligt, hvad hvis det er en større mængde instruktioner vi vil have gentaget flere gange?
Eller hvad med et variabelt antal gange?

Løkker

Vi kan gøre det med løkker!



While loop

Der er to slags løkker I vil møde det meste af tiden, og den første er `while`-løkken. Denne kører så længe en betingelse er overholdt:

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

I 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

Som tommelfingerregel så forsøg at undgå `break`.

I første omgang så forsøg at bruge den betingelse som løkken kører efter.

While loop

Som tommelfingerregel så forsøg at undgå `break`.

I første omgang så forsøg at bruge den betingelse som løkken kører efter.

I anden omgang kan følgende konstruktion bruges:

```
keeprun = True
i = 0
while(keeprun):
    print(i)
    i = i + 1
    if (i > 5):
        keeprun = False
```

While loop

Som tommelfingerregel så forsøg at undgå `break`.

I første omgang så forsøg at bruge den betingelse som løkken kører efter.

I anden omgang kan følgende konstruktion bruges:

```
keeprun = True
i = 0
while(keeprun):
    print(i)
    i = i + 1
    if (i > 5):
        keeprun = False
```

```
0
1
2
3
4
5
```

For loop

Den anden type løkke i vil møde er for-løkker.

I Python fungerer det som et for each loop. Altså for hvert element i en eller anden mængde, f.eks. heltal fra 0 til 4.

```
for <variabel> in <mængde>:
    <instruktion>
    <instruktion>
    <instruktion>
```

For loop

Den anden type løkke i vil møde er for-løkker.

I Python fungerer det som et for each loop. Altså for hvert element i en eller anden mængde, f.eks. heltal fra 0 til 4.

```
for <variabel> in <mængde>:
    <instruktion>
    <instruktion>
    <instruktion>
```

Eksempel:

```
for i in range(4):
    print(i)
```

```
0
1
2
3
```

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 en:

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

```
1
4
7
```

Nested loops

Man kan også have løkker i løkker:

```
for i in range(2):
    for j in range(3):
        print(i, j)
```

Nested loops

Man kan også have løkker i løkker:

```
for i in range(2):
    for j in range(3):
        print(i, j)
```

```
0 0
0 1
0 2
1 0
1 1
1 2
```

While loop

Lad os forbedre nogle af vores eksempler fra sidste gang!

Fibonacci (igen)

Guess the number

Lister

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 (tilgang af data) foregår med de kantede parenteser:

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

Indeksering i lister

Indeksering (tilgang af data) foregår med de kantede parenteser:

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

```
3
```

Wat?

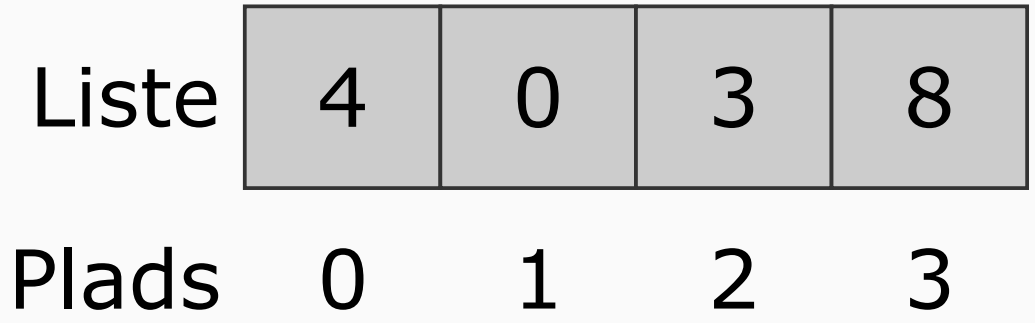
Indeksering i lister

Indeksering (tilgang af data) foregår med de kantede parenteser:

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

```
3
```

Wat? Vi tæller fra 0.



Indeksering i lister

Vi kan indeksere med alle slags værdier:

```
mylist = [4, 0, 3, 8]
x = 1
print(mylist[x])
print(mylist[len(mylist)-1])
```

Indeksering i lister

Vi kan indeksere med alle slags værdier:

```
mylist = [4, 0, 3, 8]
x = 1
print(mylist[x])
print(mylist[len(mylist)-1])
```

```
0
8
```

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 ekskluderes.

Iteration af en liste

Et gennemløb af en liste er en typisk operation:

```
mylist = [4, 0, 3, 8]
for elm in mylist:
    print(elm)
```

```
4
0
3
8
```

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 = [4, 0, 3, 8]
print(mylist)
mylist[0] = 7
print(mylist)
```

```
[4, 0, 3, 8]
[7, 0, 3, 8]
```

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

Listemetoder

Tilføj et element til en liste:

```
mylist = [4, 0, 3, 8]
print(mylist)
mylist.append(5)
print(mylist)
```

```
[4, 0, 3, 8]
[4, 0, 3, 8, 5]
```

Listemetoder

Tilføj et element til en liste:

```
mylist = [4, 0, 3, 8]
print(mylist)
mylist.append(5)
print(mylist)
```

```
[4, 0, 3, 8]
[4, 0, 3, 8, 5]
```

Fjern fra en liste

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

```
[4, 0, 3, 8]
[0, 3, 8]
```

Listemetoder

Længde af en liste.

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

Listemetoder

Længde af en liste.

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

```
4
```

Listemetoder

Længde af en liste.

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

```
4
```

Concatenation:

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

Listemetoder

Længde af en liste.

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

```
4
```

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]
```

Hmm, minder det os om noget?...

Listemetoder

Multiplication:

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

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

Hmm, minder det os om noget?...

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 (7 in yndlings):
    print("7 er nice")
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
```

Want more?

Flere metoder og eksempler kan findes her:

`https://docs.python.org/3/tutorial/datastructures.html`