



Lær Python dag 2 - modul 1

Løkker og lister

Jonas Bamse Andersen

Institut for Matematik og Datalogi - IMADA
Syddansk Universitet

1. Recap
2. Løkker
3. Lister

Recap

Alle deltagere den sidste gang kan få et certifikat på deltagelse.

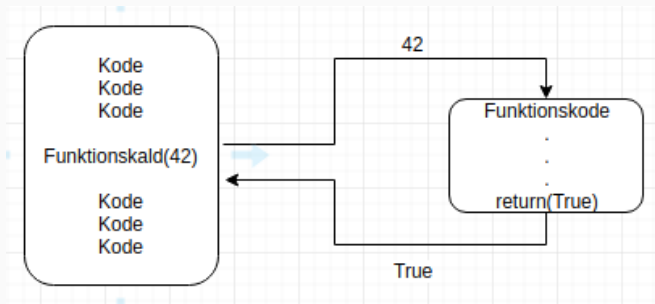


Hvad lærte I sidst?

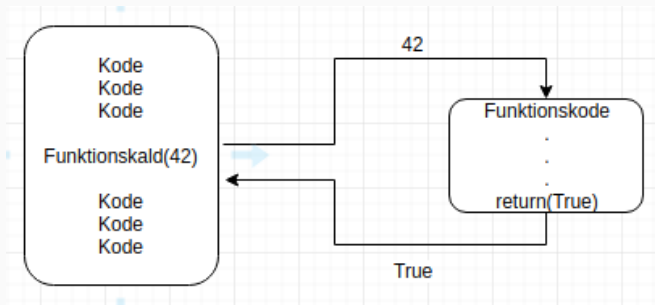
Diskutér to minutter med sidemanden, hvad I lærte sidst.

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

Apropos funktioner



Apropos funktioner



Lad os prøve at lave Fibonacci tallene.

Løkker

Tæl til 5

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

Tæl til 5

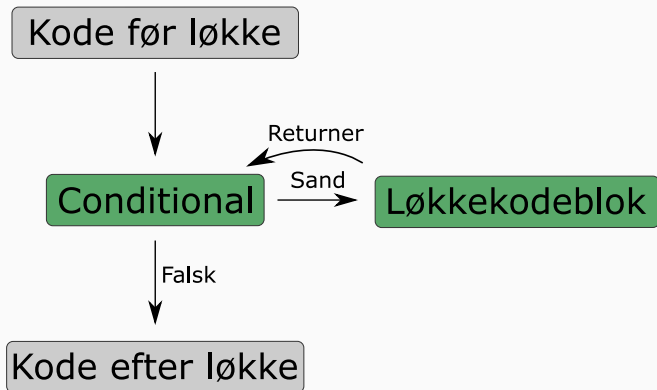
Hvordan kan vi skrive kode som tæller fra 1-5 (printer tallene)?

```
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?

Vi kan gøre det med løkker!



For loop

Gentagelse et fast antal gange ved hjælp af for-løkke:

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

For loop

Gentagelse et fast antal gange 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

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
```


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):
        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

While loop

Lad os forbedre nogle af vores eksempler fra sidste gang!

Fibonacci (igen)

Guess the number

Lister

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

- Wiki

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]
list2 = ["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

WTF?

Indeksering i lister

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

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

3

WTF? Vi tæller fra 0.

Liste	4	0	3	8
Plads	0	1	2	3

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

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

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 `for`løkker.

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.

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]
```

Længde af en liste.

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


Længde af en liste.

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

4

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]

Multiplication:

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

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

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 (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
```


Want more?

Flere metoder og eksempler kan findes her:

`https:`

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