0.1 Introduksjon til Python

```
1 print("Hello world!")
```

Output

```
Hello world!
```

Python deler talll inn i tre typer:

`int`	reelle heltall
`float`	relle tall
`complex`	komplekse tall

Vi skal i denne boka konsentrere oss om `int` og `float`. Tallypene definerer vi ved å ekskludere eller inkludere punktum:

```
1 a = 3 # a er av typen int
2 e = -5 # e er av typen int
3 b = 2.8 # b er av typen float
4 c = 2. # c=2.0, og er av typen float
5 d = .7 # d=0.7, og er av typen float
6 f = -0.01 # f er av typen float
```

```
1 a = 5
2 b = 2
3
4 print("a+b = ", a+b);
5 print("a-b = ", a-b);
6 print ("a*b = " ,a*b);
7 print ("a/b = " ,a/b);
8 print("a**b = " ,a**b); # potens med grunntall a og
                          # eksponent b
9 print ("a//b = ", a//b); # 5/2 rundet ned til nærmeste
                           # heltall
10 print ("a%b = ", a%b); # resten til a//b
```

Output

```
a+b = 7
a-b = 3
a*b = 10
a/b = 2.5
a**b = 25
a//b = 2
a%b = 1
```

Boolske verdier og vilkår

De boolske verdiene er verdiene `True` og `False`. Disse vil være resultatet når vi sjekker om objekter er like eller ulike. For å sjekke dette har vi de sammenlignende operatorene:

operator	betydning
==	er lik
!=	er *ikke* lik
>	er større enn
>=	er større enn, eller lik
<	er mindre enn
<=	er mindre enn, eller lik

```
a = 5
b = 4

print(a == b)
print(a != b)
print(a > b)
print(a < b)
```

Output

```
False
True
True
False
```

I tillegg til de sammenlignende operatorene kan vi bruke de logiske operatorene `and`, `or` og `not`

```
a = 5
b = 4
c = 9

print(a == b and c > a)
print(a == b or c > a)
print(not a == b)
```

Output

```
False
True
True
```

Språkboksen

Sjekker som bruker de sammenlignende og de logiske operatorene, skal vi heretter kalle vilkår.

`if`, `else` og `elif` uttrykkene

Når vi ønsker å utføre handlinger bare *hvis* et vilkår er sant (`True`), bruker vi `if` uttrykket foran vilkåret. Koden vi skriver med innrykk under `if`-linjen, vil bare bli utført hvis vilkåret gir `True`.

```
a = 5
b = 4
c = 9

if c > b: # legg merke til kolon (:) til slutt
  print("Jepp, c er større enn b")

if a > c: # legg merke til kolon (:) til slutt
  print("Denne teksten kommer ikke i output, siden
    vilkåret er False")
```

Output

```
Jepp, c er større enn b
```

Hvis man først vil sjekke om et vilkår er sant, og så utføre handlinger hvis det *ikke* er det, kan vi bruke `else` uttrykket:

```
a = 5
c = 9

if a > c: # legg merke til kolon (:) til slutt
  print("Denne teksten kommer ikke i output, siden
    vilkåret er False")

else: # legg merke til kolon (:) til slutt
  print("Men denne kommer, fordi vilkåret i if-linja
    over var False")
```

Output

```
Men denne kommer, fordi vilkåret i if-linja over var False
```

`else` uttrykket tar bare hensyn til (og gir ikke mening uten) `if` uttrykket like over seg. Hvis vi vil at handlinger skal utføres *bare* hvis

ingen tidligere `if` uttrykk ga noe utslag, må vi bruke[1] `elif` uttrykket. Dette er et `if` uttrykk som slår inn hvis `if` uttrykket over *ikke* ga utslag.

```
a = 2

if a > 3:
  print("Denne linja printes ikke, vilkåret er False")

elif a < 1: #Siden if uttrykket over ikke ga utslag,
              sjekkes vilkåret b < 1
  print("Denne linja printes ikke, vilkåret er False")

else:
  print("Nå er vi sikre på at 1 < b < 3")
```

Output

```
Nå er vi sikre på at 1 < b < 3
```

Lister

Lister kan vi bruke for å samle objekter. Objektene som er i listen kalles elementene til listen.

```
strings = ["98", "99", "100"]
floats = [1.7, 1.2]
ints = [96, 97, 98, 99, 100]
mixed = [1.7, 96, "100"]
empty = []
```

Elementene i lister er indekserte. Første objekt har indeks 0, andre objekt har indeks 1 og så videre:

[1] `elif` er en forkortelse for `else if`, som også kan brukes.

```python
strings = ["98", "99", "100"]
floats = [1.7, 1.2]
ints = [96, 97, 98, 99, 100]
mixed = [1.7, 96, "100"]
empty = []
```

Output

```
96
99
98
```

Med den innebygde funksjonen `append()` kan vi legge til et objekt i enden av listen. Dette er en innebygd funksjon[1], som vi skriver i enden av navnet på listen, med et punktum foran.

```python
min_liste = []
print(min_liste)

min_liste.append(3)
print(min_liste)

min_liste.append(7)
print(min_liste)
```

Output

```
[]
[3]
[3, 7]
```

Med funksjonen `pop()` kan vi hente ut et objekt fra listen

[1] Kort fortalt betyr det at det bare er noen typer objekter som kan bruke denne funksjonen.

```
min_liste = [6, 10, 15, 19]

a = min_liste.pop() # a = det siste elementet i listen
print("a =",a)
print("min_liste =",min_liste)

a = min_liste.pop(1) # a = elementet med indeks 1
print("a =",a)
print("min_liste =",min_liste)
```

Output

```
a = 19
min_liste = [6, 10, 15]
a = 10
min_liste = [6, 15]
```

Forklar for deg selv

Hva er forskjellen på å skive `a = min_liste[1]` og å skrive `a = min_liste.pop(1)`?

`for` looper

For objekter som inneholder flere elementer, kan vi bruke `for`-looper til å utføre handlinger for hvert element. Handlingene må vi skrive med et innrykk etter `for`-uttrykket:

```
min_liste = [5, 10, 15]

for number in min_liste:
  print(number)
  print(number*10)
  print("\n") # lager et blankt mellomrom

```

Output

```
5
50

10
100

15
150
```

Språkboksen

Å gå gjennom hvert element i (for eksempel) en liste kalles å iterere over listen.

Ofte er det ønskelig å iterere over heltallene $0, 1, 2$ og så videre. Til dette kan vi bruke `range()`-funksjonen:

```
ints = range(3)

for i in ints:
  print(i)

```

Output

```
0
1
2
```

`while` loop

Hvis vi ønsker at handlinger skal utføres fram til et vilkår er sant, kan vi bruke `while` metoden:

```
a = 1

while a < 5:
  print(a)
  a += 1
```

Output

```
1
2
3
4
```

Obs!

Bruk aldri innebygde funksjoner som navn på variabler.