

0.1 Introduksjon til Python

Python er et programmeringsspråk for **tekstbasert koding**. Dette innebærer at handlingene vi ønsker utført, må kodes som tekst. Filen som inneholder hele koden kaller vi et **skript**. Det synlige resultatet av å kjøre skriptet, kaller vi **utdata**¹. Det er mange måter å få kjørt skriptet sitt på, blant annet kan man bruke en online compiler som programmiz.com.

0.1.1 Objekt, type, funksjon og uttrykk

Vårt første skript består av bare én kodelinje:

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

Utdata

Hello world!

I kommende avsnitt vil begrepene **objekt**, **type**, **funksjon** og **uttrykk** stadig dukke opp.

- Det aller meste i Python er objekter. I skriptet over er både **print()** og **"Hello world"** objekter.
- Objekter vil være av forskjellige typer. **print()** er av typen **function**, mens **"Hello world"** er av typen **str**². Hvilke handlinger som kan utføres med forskjellige objekter avhenger av hvilke typer de er.
- Funksjoner kan ta imot **argumenter**, for så å utføre handlinger. I skriptet over tar **print()**-funksjonen imot argumentet **"Hello world"**, og printer teksten til utdata.
- Uttrykk har sterke likehetstrekk med funksjoner, men tar ikke imot argumenter.

¹**Output** på engelsk.

²'str' er en forkortelse for det engelske ordet 'string'.

Tilvising og utregning

Tekst og tall kan vi se på som noen av de minste byggsteinene (objektene). Python har én type for tekst, og to typer for reelle tall:

<code>str</code>	tekst
<code>int</code>	heltall
<code>float</code>	desimaltall

Det er som regel nyttig å gi objektene våre navn. Dette gjør vi ved å skrive navnet etterfulgt av `=` og objektet. **Kommentarer** er tekst som ikke blir behandlet som kode. Kommentarer kan vi skrive ved å starte setningen med `#`.

```
1 hei = "hei" # hei er av typen str. Legg merke til "  
                ved start og slutt  
2 a = 3 # a 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 e = -5 # e er av typen int  
7 f = -0.01 # f er av typen float
```

Med Python kan vi selvsagt utføre klassiske regneoperasjoner:

```
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); # a/b rundet ned til nærmeste  
                          heltall  
10 print("a%b = ", a%b); # resten til a//b
```

Utdata

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

Funksjonene `str()`, `int()` og `float()` kan vi bruke til å gjøre om objekter til typene `int` eller `float`:

```
1 s = "2"
2 b = 3
3 c = 2.0
4
5 b_s = str(b) # b omgjort til str
6 c_s = str(c) # c omgjort til str
7 print(b_s+c_s)
8
9 s_i = int(s)
10 print(s_i*b)
11
12 s_f = float(s)
13 print(s_f*b)
```

Utdata

32.0

6

6.0

En viktig ting å være klar over er at `=` i Python *ikke* betyr det samme som `=` i matematikk. Mens `=` kan oversettes til "er lik", kan vi si at `=` kan oversettes til 'er'.

```
1 a = 5 # a ER nå 5
2 print(a)
3 a = a+1 # a ER nå det a VAR, + 1
4 print(a)
```

Utdata

5

6

At et objekt legger til seg selv og en annen verdi er så vanlig i programmering at Python har en egen operator for det:

```
1 a = 5 # a ER nå 5
2 a+= 1 # Samme som å skrive a = a+1
3 print(a)
```

Utdata

5
6

Selv om datamaskiner er ekstremt raske til å utføre utregninger, har de en begrensning det er viktig å være klar over; avrundingsfeil. En av grunnene til dette er at datamaskiner bare kan bruke et visst antall desimaler for å representere tall. En annen grunn er at datamaskiner anvender [totalssystemet](#). Det er mange verdier vi kan skrive eksakt i titalssystemet som ikke lar seg skrive eksakt i totalssystemet. For å bøte på dette kan vi bruke funksjonen `round()`:

```
1 a = 8.3/10
2 print(a) # avrundingsfeil, da vi skulle hatt a = 0.83
3
4 a = round(a, 2) # runder av a til tall med to
   desimaler
5 print(a)
```

Utdata

0.8300000000000001
0.83

0.1.2 Egne funksjoner

I Python kan man enkelt lage sine egne funksjoner. En funksjon kan utføre handlinger, og den kan **returnere** (return på engelsk) ett eller flere objekt. Den kan også ta imot argumenter. Koden vi skriver inni en funksjon blir bare utført hvis vi **kaller** (call på engelsk) på funksjonen.

```
1 # a er en funksjon som ikke tar noen argumenter.
2 # Legg merke til 'def' først og ':' til slutt.
3 # Kodelinjene som hører til funksjonen må stå med
   innrykk
4 def a():
5     print("Hei, noen kalte visst på funksjon a?")
6
7
8 # b er en funksjon som tar argumentet 'test'
9 def b(tekst):
10    print("Hei. Noen kalte på funksjon b. Argumentet som
       ble gitt var: ", tekst)
11
12 # c er en funksjon som tar argumentene a og b
13 # c returnerer et objekt
14 def c(a, b):
15     return a+b
16
17 b("Hello!") # vi kaller på b med argumentet "hello"
18
19 d = c(2,3) # Vi kaller på a med argumentene 2 og 3
20
21 print(d)
22
23 # merk at teksten gitt i a ikke blir printet, fordi vi
       ikke har kalt på a.
24
25
26
27
28
29
```

Utdata

Hei. Noen kalte på funksjon b. Argumentet som ble gitt var:
Hello!

5

0.1.3 Boolske verdier og vilkår

Verdiene **True** og **False** kalles **boolske verdier**. Disse vil være resultatet når vi sjekker om objekter er like eller ulike. For å sjekke dette har vi de **sammenlignende operatorene**:

operator	betydning
<code>==</code>	er lik
<code>!=</code>	er <i>ikke</i> lik
<code>></code>	er større enn
<code>>=</code>	er større enn, eller lik
<code><</code>	er mindre enn
<code><=</code>	er mindre enn, eller lik

```
1 a = 5
2 b = 4
3
4 print(a == b)
5 print(a != b)
6 print(a > b)
7 print(a < b)
```

Utdata

```
False
True
True
False
```

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

```
1 a = 5
2 b = 4
3 c = 9
4
5 print(a == b and c > a)
6 print(a == b or c > a)
7 print(not a == b)
```

Utdata

```
False
True
True
```

Språkboksen

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

0.1.4 Uttrykkene **if**, **else** og **elif**

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

```
1 a = 5
2 b = 4
3 c = 9
4
5 if c > b: # legg merke til kolon (:) til slutt
6     print("Jepp, c er større enn b")
7
8 if a > c: # legg merke til kolon (:) til slutt
9     print("Denne teksten kommer ikke i output, siden
      vilkåret er False")
```

Utdata

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 uttrykket **else**:

```
1 a = 5
2 c = 9
3
4 if a > c: # legg merke til kolon (:) til slutt
5     print("Denne teksten kommer ikke i output, siden
      vilkåret er False")
6
7 else: # legg merke til kolon (:) til slutt
8     print("Men denne kommer, fordi vilkåret i if-linja
      over var False")
```

Utdata

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

Uttrykket **else** 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 bruke¹ uttrykket `elif`. Dette er et `if`-uttrykk som slår inn hvis `if`-uttrykket over *ikke* ga utslag.

```
1 a = 2
2
3 if a > 3:
4     print("Denne linja printes ikke, vilkåret er False")
5
6 elif a < 1: #Siden if uttrykket over ikke ga utslag,
7             sjekkes vilkåret b < 1
8             print("Denne linja printes ikke, vilkåret er False")
9
10 else:
11     print("Nå er vi sikre på at 1 < b < 3")
```

Utdata

Nå er vi sikre på at $1 < b < 3$

Merk

Når du jobber med tall, kan noen vilkår du forventer skal være `True` vise seg å være `False`. Dette handler ofte om avrundingsfeil, som vi har omtalt på side 4.

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

0.1.5 Lister

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

```
1 strings = ["98", "99", "100"]
2 floats = [1.7, 1.2]
3 ints = [96, 97, 98, 99, 100]
4 mixed = [1.7, 96, "100"]
5 empty = []
```

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

```
1 strings = ["98", "99", "100"]
2 floats = [1.7, 1.2]
3 ints = [96, 97, 98, 99, 100]
4 mixed = [1.7, 96, "100"]
5 empty = []
```

Utdata

```
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.

```
1 min_liste = []
2 print(min_liste)
3
4 min_liste.append(3)
5 print(min_liste)
6
7 min_liste.append(7)
8 print(min_liste)
```

Utdata

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

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

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

```
1 min_liste = [6, 10, 15, 19]
2
3 a = min_liste.pop() # a = det siste elementet i listen
4 print("a =",a)
5 print("min_liste =",min_liste)
6
7 a = min_liste.pop(1) # a = elementet med indeks 1
8 print("a =",a)
9 print("min_liste =",min_liste)
```

Utdata

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

Forklar for deg selv

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

Med funksjonen `sort()` kan vi sortere elementene i listen.

```
1 heltall = [9, 0, 8, 3, 1, 7, 4]
2 bokstaver = ['c', 'a', 'b', 'e', 'd']
3
4 heltall.sort()
5 bokstaver.sort()
6
7 print(heltall)
8 print(bokstaver)
```

Utdata

```
[0, 1, 3, 4, 7, 8, 9]
['a', 'b', 'c', 'd', 'e']
```

Med funksjonene `count()` kan vi telle gjentatte elementer i listen.

```
1 heltall = [2, 7, 2, 2, 2]
2 frukt = ['banan', 'eple', 'banan']
3
4 antall_toere = heltall.count(2)
5 antall_sjuere = heltall.count(7)
6 antall_bananer = frukt.count('banan')
7 antall_appelsiner = frukt.count('appelsin')
8
9 print(antall_toere)
10 print(antall_sjuere)
11 print(antall_bananer)
12 print(antall_appelsiner)
```

Utdata

4
1
2
0

Med funksjonen `len()` kan vi finne antall elementer i en liste, og med funksjonen `sum()` kan vi finne summen av lister med tall som elementer.

```
1 heltall = [2, 7, 2, 2, 2]
2 frukt = ['banan', 'eple', 'banan']
3
4 print(len(heltall))
5 print(len(frukt))
6 print(sum(heltall))
```

Utdata

5
3
15

Med uttrykket `in` kan vi sjekke om et element er i en liste.

```
1 heltall = [1, 2, 3]
2
3 print(1 in heltall)
4 print(0 in heltall)
```

Utdata

True

False

0.1.6 Looper; **for** og **while**

for loop

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:

```
1 min_liste = [5, 10, 15]
2
3 for number in min_liste:
4     print(number)
5     print(number*10)
6     print("\n") # lager et blankt mellomrom
7
```

Utdata

```
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()**:

```
1 ints = range(3)
2
3 for i in ints:
4     print(i)
5
```

Utdata

```
0
1
2
```

while loop

Hvis vi ønsker at handlinger skal utføres fram til et vilkår er sant, kan vi bruke en **while**-loop:

```
1 a = 1
2
3 while a < 5:
4     print(a)
5     a += 1
```

Utdata

```
1
2
3
4
```