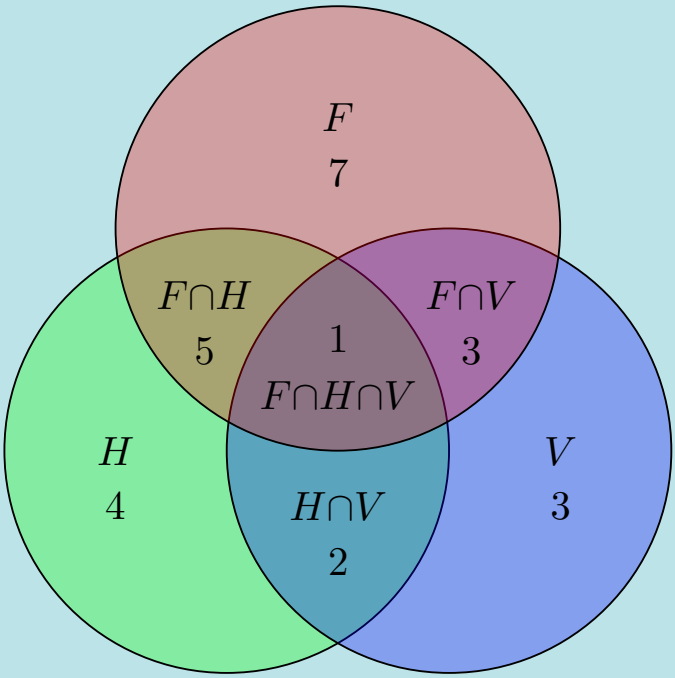
Matematikk P1



Sindre Sogge Heggen

Innhold

Kapittel 1

Digitale verktøy

En viktig del av å beherske digitale verktøy, er å forstå grunnleggende **programmering**. Programmering handler om å gi instrukser til en datamaskin. Slik kan datamaskiner utføre utregninger, framstille bilder, animasjoner, spill, og mye mer. For å gi instrukser bruker vi forskjellige **programmeringsspråk**, og det er et hav av forskjellige språk å velge i. I norsk skole er de de mest brukte språkene Scratch, Python og JavaScript[2]. Det finnes et stort utvalg av gratis ressurser for å lære seg programmeringsspråk, blant andre

- code.org (koding generelt)
- w3schools.com (koding generelt)
- scratch.mit.edu (Scratch)
- microbit.org (koding med micro:bit)
- espensklasserom.co (Koding i Sracht, micro:bit m.m.)
- kidsakoder.no (koding i Scratch, micro:bit, Python m.m)

Har du allerede nådd et høyt nivå som programmerer, og føler du har god kontroll på data-typer, funksjoner, klasser o.l.? Da anbefales språket Rust. Mange holder dette for å være arvtakeren til C++ og liknende språk.

Innhold

[2]Rett nok i blokkbasert utgave ved koding av micro:bit.

1.1 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.

1.1.1 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 byggesteinene (objektene). Python har én type for tekst, og to typer for reelle tall:

`str`	tekst
`int`	heltall
`float`	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 `#`.

```
hei = "hei" # hei er av typen str. Legg merke til "
              ved start og slutt
a = 3 # a er av typen int
b = 2.8 # b er av typen float
c = 2. # c=2.0, og er av typen float
d = .7 # d=0.7, og er av typen float
e = -5 # e er av typen int
f = -0.01 # f er av typen float
```

Med Python kan vi selvsagt utføre klassiske regneoperasjoner:

```
a = 5
b = 2

print("a+b = ", a+b);
print("a-b = ", a-b);
print ("a*b = " ,a*b);
print ("a/b = " ,a/b);
print("a**b = " ,a**b); # potens med grunntall a og
                          eksponent b
print ("a//b = ", a//b); # a/b rundet ned til nærmeste
                          heltall
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:

```
a = 5 # a ER nå 5
a+= 1 # Samme som å skrive a = a+1
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 totallssystemet. Det er mange verdier vi kan skrive eksakt i titallssystemet som ikke lar seg skrive eksakt i totallssystemet. For å bøte på dette kan vi bruke funksjonen `round()`:

```
a = 8.3/10
print(a) # avrundingsfeil, da vi skulle hatt a = 0.83

a = round(a, 2) # runder av a til tall med to
    desimaler
print(a)
```

Utdata

```
0.8300000000000001
0.83
```

1.1.2 Egne funksjoner

Ved å bruke metoden `def` kan man 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.

```
# a er en funksjon som ikke tar noen argumenter.
# Legg merke til 'def' først og ':' til slutt.
# Kodelinjene som hører til funksjonen må stå med innrykk
def a():
  print("Hei, noen kalte visst på funksjon a?")


# b er en funksjon som tar argumentet 'test'
def b(tekst):
  print("Hei. Noen kalte på funksjon b. Argumentet som ble gitt var: ", tekst)

# c er er funksjon som tar argmunentene a og b
# c returnerer et objekt
def c(a, b):
  return a+b

b("Hello!") # vi kaller på b med argumentet "hello"

d = c(2,3) # Vi kaller på a med argumentene 2 og 3

print(d)

# merk at teksten gitt i a ikke blir printet, fordi vi ikke har kalt på a.
```

Utdata

```
Hei. Noen kalte på funksjon b. Argumentet som ble gitt var:
Hello!
5
```

1.1.3 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
==	er lik
!=	er *ikke* lik
>	er større enn
>=	er større enn, eller lik
<	er mindre enn
<=	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**.

1.1.4 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`.

```
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")
```

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`:

```
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")
```

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 bruke[1] uttrykket `elif`. 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")
```

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 6.

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

1.1.5 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:

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

```
min_liste = []
print(min_liste)

min_liste.append(3)
print(min_liste)

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

Utdata

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

[1] Kort 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

```
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)
```

Utdata

```
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)`?

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

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

heltall.sort()
bokstaver.sort()

print(heltall)
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.

```
heltall =  [2, 7, 2, 2, 2]
frukt = ['banan', 'eple', 'banan']

antall_toere = heltall.count(2)
antall_sjuere = heltall.count(7)
antall_bananer = frukt.count('banan')
antall_appelsiner = frukt.count('appelsin')

print(antall_toere)
print(antall_sjuere)
print(antall_bananer)
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.

```
heltall =  [2, 7, 2, 2, 2]
frukt = ['banan', 'eple', 'banan']

print(len(heltall))
print(len(frukt))
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
```

1.1.6 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

1.1.7 `input()`

Vi kan bruke funksjonen `input()` til å skrive inn tekst mens skriptet kjører:

```
1 innskrevet_tekst = input("Skriv inn her: ")
2 print(innskrevet_tekst)
```

Teksten vi har skrevet inni `input()` i skriptet over er teksten vi ønsker vist foran teksten som ønskes innskrevet. Linje 2 i denne koden vil ikke kjøres før en tekst er innskrevet.

```
1 innskrevet_tekst = input("Skriv inn her: ")
2 print(innskrevet_tekst)
```

Utdata
```
Skriv inn tekst her: OK
OK
```

Objektet gitt av en `input()`-funksjon vil alltid være av typen `str`. Man må alltid passe på å gjøre om objekter til rett type:

```
print("La oss regne ut a*b")
a_str = input("a = ")
b_str = input("b = ")
a = float(a_str)
b = float(b_str)
print("a*b = ", a*b)
```

Utdata

```
La oss regne ut a*b
a = 3.7
b = 4
a*b = 14.8
```

1.1.8 Moduler

Noen ganger er vi nødt til å importere **moduler** for å få tilgang til objekter vi ønsker. Dette gjør vi ved å bruke `import`-metoden:

```
# importerer konstanten pi fra modulen 'math'
from math import pi
print(pi)

# importerer funksjonen randint fra modulen 'random'
from random import randint

# randint(2, 5) returnerer tilfeldig valgt heltall
# mellom (og inkludert) 2 og 5.
print(randint(2, 5))
```

Utdata

```
3.141592653589793
4
```

1.1.9 Feilmeldinger

Påstand: Alle programmerere vil erfare at skriptet ikke kjører fordi vi ikke har skrevet koden på rett måte. Dette kalles en **syntax error**. Ved syntax error vil man få beskjed om på hvilken linje feilen befinner seg, og hva som er feil. De vanligste feilene er

- **Å glemme innrykk når man bruker metoder som `def`, `for`, `while`, og `if`**

```
1 a = 472
2 b = 98
3
4 if a*b > 48000:
5 print("a*b er større enn 48000")
```

Utdata

```
line 5, in <module>
print("a*b er større enn 48000")
^
IndentationError: expected an indented block after
'if' statement on line 4
```

- **Å utføre operasjoner på typer det ikke gir mening for**

```
1 b = "98"
2 b_opphøyd_i_andre = b**2
```

Utdata

```
line 2, in <module>
b_opphøyd_i_andre = b**2
TypeError: unsupported operand type(s) for ** or
pow(): 'str' and 'int'
```

Oppgaver for kapittel 1

1.1.1

Lag en funskjon `f(a, b)` som

a) printer $a \cdot b$.

b) printer a^2+b^2. Hva tilsvarer dette tallet hvis a og b er lengdene til katetene i en rettvinklet trekant?

c) runder av et desimaltall a til et tall med b desimaler.

d) printer om a er større, lik, eller mindre enn b.

1.1.2

Skriv outputen til dette skriptet:

```
1 for i in range(4):
2   print(i + 2)
```

1.1.3

Skriv outputen til dette skriptet:

```
1 i = 0
2 while i < 5:
3   i = i + 2
4   print(i)
```