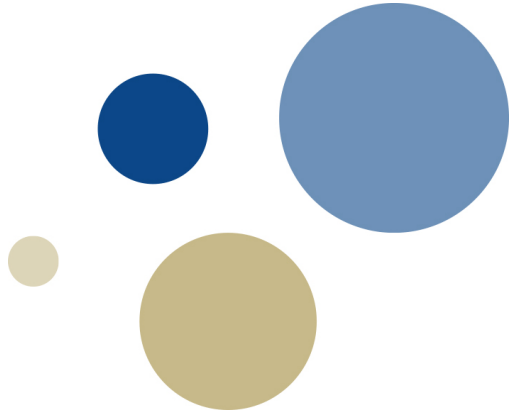
NTNU

Kunnskap for en bedre verden



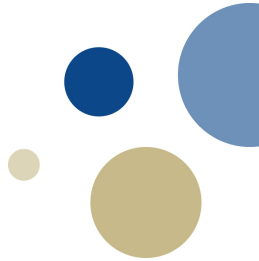
Kræsjkurs

aaa

Ola Nordmann
, NTNU

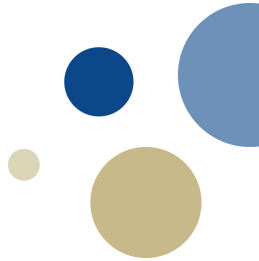
1 January 1970

Tilordningsoperatoren



Gitt to variabler *a* og *b*, bytt om innholdet.
Tilordningsoperatoren =
Original verdi blir overskrevet.

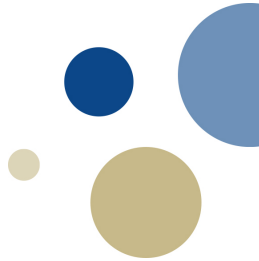
Tilordningsoperatoren



```
1  a = "hei"
2  b = 3.14
3  a = b
4  b = a
```

	1	2	3	4
a				
b				

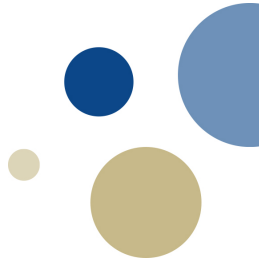
Tilordningsoperatoren



```
1  a = "hei"
2  b = 3.14
3  a = b
4  b = a
```

	1	2	3	4
a	"hei"			
b	?			

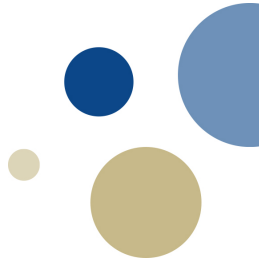
Tilordningsoperatoren



```
1  a = "hei"
2  b = 3.14
3  a = b
4  b = a
```

	1	2	3	4
a	"hei"	"hei"		
b	?	3.14		

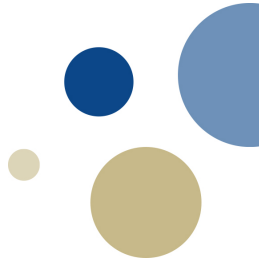
Tilordningsoperatoren



```
1  a = "hei"
2  b = 3.14
3  a = b
4  b = a
```

	1	2	3	4
a	"hei"	"hei"	3.14	
b	?	3.14	3.14	

Tilordningsoperatoren

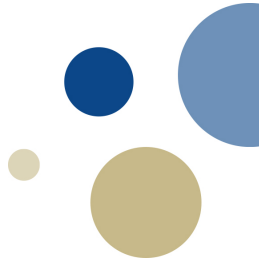


Problem: Gammel verdi blir overskrevet. Vi kan bare endre en ting av gangen.

```
1  a = "hei"
2  b = 3.14
3  a = b
4  b = a
```

	1	2	3	4
a	"hei"	"hei"	3.14	3.14
b	?	3.14	3.14	3.14

Tilordningsoperatoren

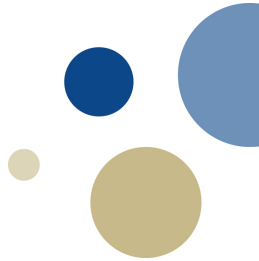


To vanlige løsninger

```
1  a = "hei"
2  b = 3.14
3  old_a = a    # ofte kalt temporary / temp
4  a = b
5  b = old_a
```

	1	2	3	4	5
a	"hei"	"hei"	"hei"	3.14	3.14
b	?	3.14	3.14	3.14	"hei"
a_old	?	?	"hei"	"hei"	"hei"

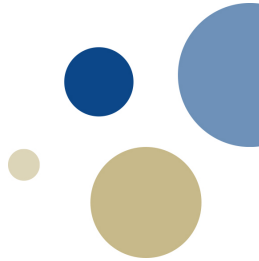
Tilordningsoperatoren



To vanlige løsninger

```
1  a = "hei"
2  b = 3.14
3  (b, a) = (a, b)
4  # bruker tupler
5  # kommer tilbake til dette senere
```

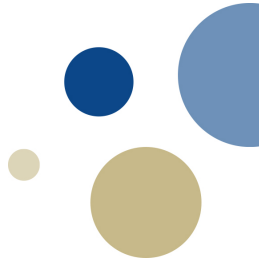
Tilordningsoperatoren



Flere tilordninger etter hverandre

```
1   a = b = c = 3.14
```

Tilordningsoperatoren



Flere tilordninger etter hverandre

Triks: Parenteser.

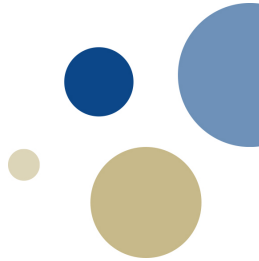
```
1  a = (b = (c = 3.14))
2  # ikke riktig syntaks, kun for illustrere
```

Flere tilordningsoperatorer

Finnes flere tilordningsoperatorer.
VIKTIG: Bortsett fra standard tilordning, må variabelen som tilordnes har en verdi fra før av.

Operator	Eksempel	Det samme som
=	a = 1	a = 1
-=	a -= 1	a = a - 1
/=	a /= 1	a = a / 1
//=	a //= 1	a = a // 1
+=	a += 1	a = a + 1
*=	a *= 1	a = a * 1
%=	a %= 1	a = a % 1
**=	a **= 1	a = a**1

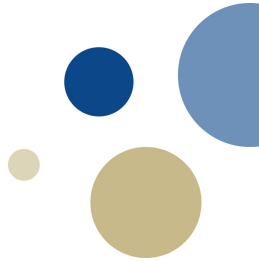
Aritmetiske operatorer



Viktig: Holde tunga rett i munnen med //, % og **

Operator	Eksempel
+	1 + 2
-	1 - 2
/	1 / 2
//	1 // 2
*	1 * 2
%	1 % 2
**	1**2

Aritmetiske operatorer



Hva skjer her?

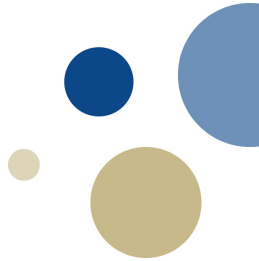
```
1  from math import sqrt
2  a = sqrt(2)
3  b = 2**(1/2)
4
5  print(a == b)
```

Aritmetiske operatorer

Hva skjer her?
True printes. Hvorfor trenger man da *sqrt* funksjonen?

```
1  from math import sqrt
2  a = sqrt(2)
3  b = 2**(1/2)
4
5  print(a == b)
```

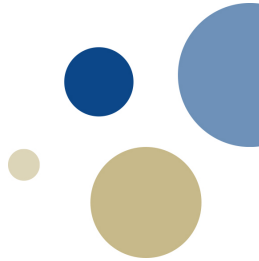
Datatyper



Variabler har bestemte typer, men kan byttes med tilordningsoperatoren.

```
1  a = "hei" # type er str
2  a = 1 # type er int
```

Datatyper



Noen aritmetiske operatorer fungerer for andre typer enn tall, men ikke alltid.

```
1  liste1 = [1, 2, 3]
2  liste2 = [4, 5]
3  liste3 = liste1 + liste2
4
5  liste4 = liste3 * 2
```

Datatyper

Eksempler på datatyper:

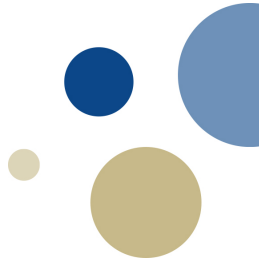
- int: heltall, 1023913213, 0, -1
- float: flyttall, 0.1, 1.2e-10, 3.1415e6
- str: streng, "hei", "sacasdcasdc"
- list: liste, [1, 2, 3], ["hei", 1.3e2]
- tuple: tupler, (1, 2, 3), ("hei", 1.3e2)

Datatyper

Funksjoner for å konvertere mellom datatyper. De gjør forskjellige ting for forskjellig input-typer, og fungerer ikke for alle.

Funksjon	Eksempel	Resultat
bin()	bin(92)	’0b1011100’
bool()	bool(12), bool(0)	True, False
chr()	chr(97)	’a’
ord()	ord(’a’)	97
float()	float(1)	1.0
int()	int(2.6), int("2")	2, 2
list()	list(range(2))	[0, 1]
str()	str(100)	’100’

Datatyper



Type gir hvilken datatype en variabel er, dermed kan vi sjekke med en if. Kan også bruke *isinstance*

```
1  a = 1.23
2  if type(a) == float:
3      print("a er en float")
4  if isinstance(a, float):
5      print("a er fortsatt en float")
```

Datatyper

Dette kan ofte være nyttig i funksjoner for å gjøre forskjellige ting for forskjellige parametere.

Input og output

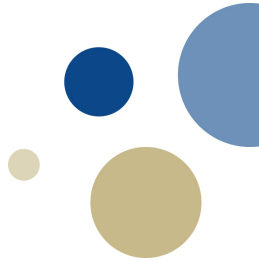
Hva er input og output?

Input og output

Hva er input og output?

— Kommunikasjon mellom bruker og kode

Input og output



Hva er input og output?

- Kommunikasjon mellom bruker og kode
- Kommunikasjon mellom datamaskin og kode

Input og output

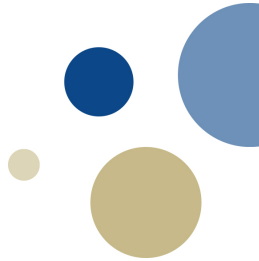
Hva er input og output?

- Kommunikasjon mellom bruker og kode: *input* and *print*
- Kommunikasjon mellom datamaskin og kode: *open*

Input og output - print

print-funksjonen.

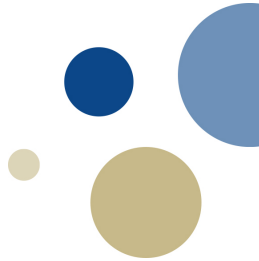
Input og output - print



print-funksjonen.

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

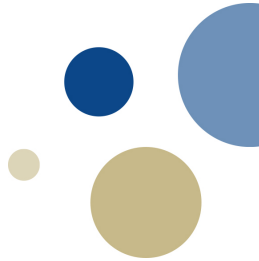
Input og output - print



print-funksjonen. Skriver disse ut det samme?

```
1  print("Hello World!")
2  print("Hello ", "World!")
```

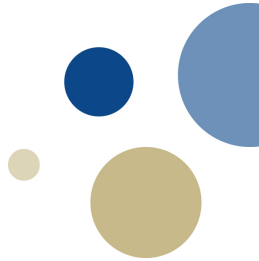
Input og output - print



print-funksjonen. Skriver disse ut det samme?
Default parameter: *sep*

```
1  print("Hello World!", sep=" ")
2  print("Hello ", "World!", sep=" ")
```

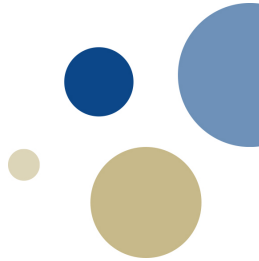
Input og output - print



print-funksjonen. Skriver disse ut det samme?
Default parameter: *sep*
Kan brukes til å endre hva som printes mellom argumentene i print-funksjonen.

```
1  print("Hello World!", sep=" ")
2  print("Hello ", "World!", sep="")
```

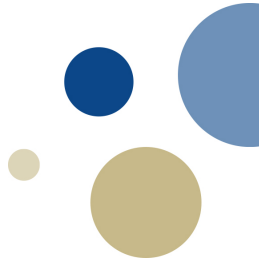
Input og output - print



print-funksjonen. Skriver disse ut det samme?

```
1  print("Hello World!")
2  print("Hello ")
3  print("World!")
```

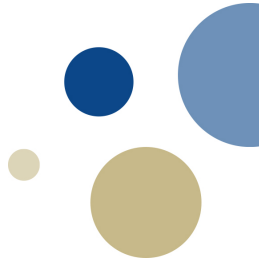
Input og output - print



print-funksjonen. Skriver disse ut det samme?
Default parameter: *end*

```
1  print("Hello World!", end="\n")
2  print("Hello ", end="\n")
3  print("World!", end="\n")
```

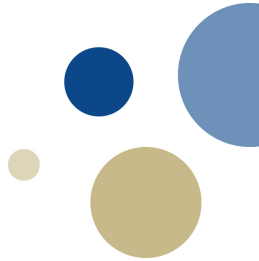
Input og output - print



print-funksjonen. Skriver disse ut det samme?
Default parameter: *end*

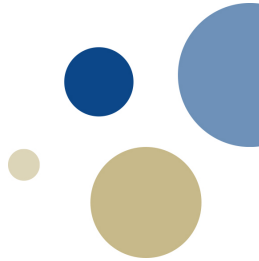
```
1  print("Hello World!", end="\n")
2  print("Hello ", end="")
3  print("World!", end="\n")
```

Input og output - print



sep og *end* er ikke alltid nødvendig å bruke, men kan gjøre koding lettere, og gjøre det enklere å formatere kode og plassere tekst.

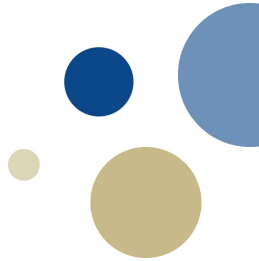
Input og output - input



input-funksjonen

```
1   x = input("Skriv inn x")
```

Input og output - input



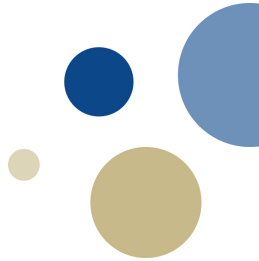
input-funksjonen.
Viktig: Tar inn en (1) streng og returnerer en (1) streng! Argumenter kan ikke skrives inn som i print, vi må bruke streng-konkatenering (plusse sammen strenger).

```
1  antall_heltall = 1
2  x = input("Skriv inn " + str(antall_heltall) + " x: ")
3  x_heltall = int(x)
```

Input og output - Lese filer

open-funksjonen brukes til å åpne filer generelt. Dette gjelder både lesing av og skriving til filer.

Input og output - Lese filer

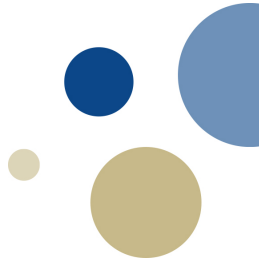


open-funksjonen brukes til å åpne filer generelt. Dette gjelder både lesing av og skriving til filer.
Vi begynner med å lese fra filer som allerede eksisterer.

Input og output - Lese filer

open-funksjonen tar inn to parametere. Begge er strenger, den første er hvilken fil du vil åpne, og den andre er hvilken *modus* du vil åpne filen i. For å lese fra filer bruker vi "r" for å lese.
Vi kan videre bestemme om vi vil åpne fila som en tekstfil eller binærfil.
Det vil si, "rt" og "rb", men "r" tolkes som "rt", siden det er vanligst, og med mindre det står eksplisitt på eksamen at man skal lese fila som en binærfil, er det "r" som mest sannsynlig skal brukes for å lese filer.

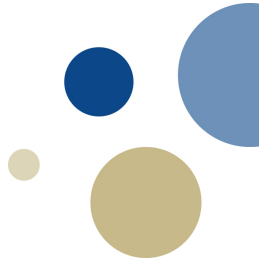
Input og output - Lese filer



Filen "test.txt" må eksistere, ellers får vi en feilmelding. I tillegg må den ligge i **samme mappe som du kjører Python-fila fra**.

```
1  f = open("test.txt", "r")
2  print(f.read())
3  f.close()
```

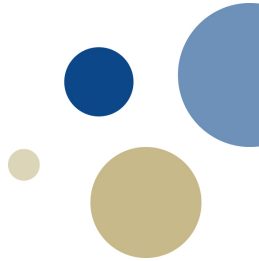
Input og output - Lese filer



Filen "test.txt" må eksistere, ellers får vi en feilmelding. I tillegg må den ligge i **samme mappe som du kjører Python-fila fra**.
Dette kalles relativ filplassering.

```
1  f = open("test.txt", "r")
2  print(f.read())
3  f.close()
```

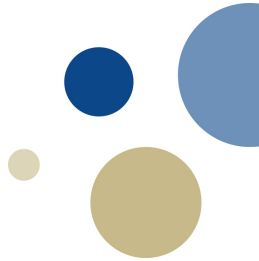
Input og output - Lese filer



Metoden *read()* henter ut **all** teksten fra fila.

```
1  f = open("test.txt", "r")
2  print(f.read())
3  f.close()
```

Input og output - Lese filer



Metoden *read()* henter ut **all** teksten fra fila. Det vil si at vi ikke kan lese to ganger fra samme objekt, og eventuelt må åpne fila på nytt.

```
1  f = open("test.txt", "r")
2  print(f.read())
3  f.close()
```

Input og output - Lese filer

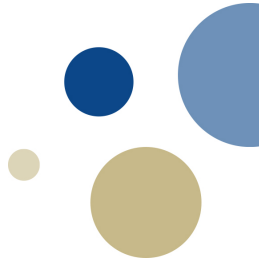
Metoden *read()* henter ut **all** teksten fra fila. Det vil si at vi ikke kan lese to ganger fra samme objekt, og eventuelt må åpne fila på nytt. Husk å lukke fila til slutt. Dette er ikke like viktig når man leser fra en fil som når man skriver til en fil, men er lurt siden man kan miste poeng på eksamen.

```
1  f = open("test.txt", "r")
2  print(f.read())
3  f.close()
```

Input og output - Lese filer

Det finnes andre metoder også, for eksempel, dersom tekstfila er for stor til å åpnes i minnet.

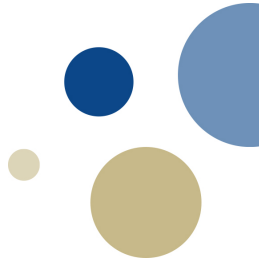
Input og output - Lese filer



Det finnes andre metoder også, for eksempel, dersom tekstfila er for stor til å åpnes i minnet.
Metoden *readline()* leser en og en linje, og når alt er lest ut returnerer den bare tomme strenger.

```
1  f = open("test.txt", "r")
2
3  f_lesing = f.readline()
4  print(f_lesing)
5  while f_lesing != "":
6      f_lesing = f.readline()
7      print(f_lesing)
```

Input og output - Lese filer



Det finnes andre metoder også, for eksempel, dersom tekstfila er for stor til å åpnes i minnet.
Metoden *readlines()* er veldig lik, men leser ut alle linjene i fila, og legger dem i en liste.
Begge disse kan være nyttige (spesielt *readlines()*) dersom man skal gjøre noe pr. linje, og kan gjøre det mer intuitivt å jobbe med filer.

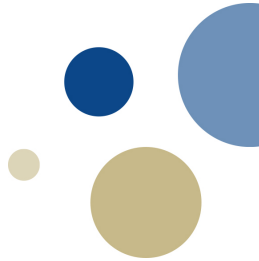
```
1  f = open("test.txt", "r")
2
3  f_linjer = f.readlines()
4  for linje in f_linjer:
5      print(linje)
```

Input og output - Lese filer

Det finnes andre metoder også, for eksempel, dersom tekstfila er for stor til å åpnes i minnet.
Begge disse kan være nyttige (spesielt *readlines()*) dersom man skal gjøre noe pr. linje, og kan gjøre det mer intuitivt å jobbe med filer.

```
1  f = open("test1.txt", "r")
2  g = open("test2.txt", "r")
3
4  f_lesing = f.readline()
5  print(f_lesing)
6  while f_lesing != "":
7      f_lesing = f.readline()
8      print(f_lesing)
9
10 g_linjer = g.readlines()
11 for linje in g_linjer:
12     print(linje)
```

Input og output - Skrive til filer



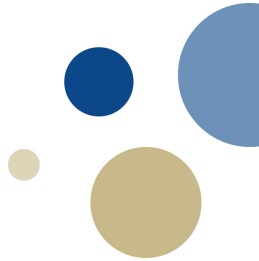
Igjen brukes *open*-funksjonen. Viktig å holde tunga rett i munnen, siden det er mye som kan gå galt her.

Input og output - Skrive til filer

Igjen brukes *open*-funksjonen. Viktig å holde tunga rett i munnen, siden det er mye som kan gå galt her.
Her er det tre nye moduser, ikke bare *"r"*, som tidligere.

- "w": Write, standard skriving. Dersom fila eksisterer fra før av, slettes den opprinnelige fila.
- "a": Append, legger til tekst på slutten av fila. Dersom den ikke eksisterer, opprettes den.
- "x": Exclusive write, vil feile dersom fila allerede eksisterer. Kan bare brukes til å opprette nye filer.

Input og output - Skrive til filer

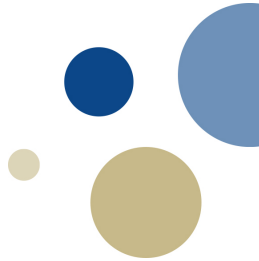


Igjen brukes *open*-funksjonen. Viktig å holde tunga rett i munnen, siden det er mye som kan gå galt her.
Her er det tre nye moduser, ikke bare *"r"*, som tidligere.

- "w": Write, standard skriving. Dersom fila eksisterer fra før av, slettes den opprinnelige fila.
- "a": Append, legger til tekst på slutten av fila. Dersom den ikke eksisterer, opprettes den.
- "x": Exclusive write, vil feile dersom fila allerede eksisterer. Kan bare brukes til å opprette nye filer.

Det finnes også en modus "+", som tillater oppdatering (read og write samtidig), men er oftest bedre å gjøre dette i to steg.

Input og output - Skrive til filer

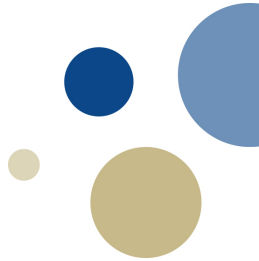


Igjen brukes *open*-funksjonen. Viktig å holde tunga rett i munnen, siden det er mye som kan gå galt her.
Tilsvarende som med å lese fra filer, kan filer også åpnes som binærfiler.

Input og output - Skrive til filer

Igjen brukes *open*-funksjonen. Viktig å holde tunga rett i munnen, siden det er mye som kan gå galt her.
Tilsvarende som med å lese fra filer, kan filer også åpnes som binærfiler.

Input og output - Skrive til filer



Skriving til fil er dermed ganske rett frem. Man skriver fra *der man er* i fila.

```
1  f = open("test.txt", "w")
2  f.write("Hello\n")
3  f.write("World!\n")
4  f.close()
5
6  g = open("test.txt", "a")
7  g.write("New\n")
8  g.write("lines!\n")
9  g.close()
```