

Velkommen!



johannph på mattermost
johannph@uio.no på mail!

DU SKAL TIL ENHVER TID...

Ha god
håndhygiene,
vasking og
antibac

Holde
avstand = 1 m

... Og føler du
deg syk skal
du gå
hjem/holde
deg hjemme

HVIS JEG FØLER MEG SYK

**GA HJEM
UMIDDEL-
BART /
HOLD DEG
HJEMME**

**Kontakt
helsevesenet
gjennom fastlege
/ koronatelefon –
de avklarer om
testing er
nødvendig**

**UiO er i
kontakt med
helsevesenet
og omvendt**

Om meg - Johanna

- Har en bachelor i statsvitenskap
- Gått to år informatikk prosa
- Går nå master prosa
- Kunne ikke noe om programmering før jeg startet
- Hadde P-matte på vgs (jeg ville heller spille candy crush) og ingen andre realfag.
- Har veldig høydeskrekk
- Elsker reality (selling sunset, anyone?)
- Send meg en melding på mattermost når som helst hvis du trenger hjelp med noe :)



Hvem skal jeg kontakte?

Spørsmål om faget:

Gruppelæreren din eller foreleser.

Studieinfo: spørsmål, klager, utsettelse av frister

<https://www.mn.uio.no/ifi/studier/kontakt/>

UiO forvei: bekymringer, noen å snakke med

<https://www.mn.uio.no/studier/forvei/>

SiO: fysisk og psykisk helsehjelp

<https://www.sio.no/helse>

<https://www.sio.no/helse/noen-%C3%A5-snakke-med>

Tips og trix

Gjør oppgaver helt til dere synes det er lett

In1000 er kjempeviktig for å forstå in1010 (som dere kanskje skal ta neste semester)

Gjør trix-oppgavene!

Ting bygger på hverandre, programmering krever innsats over tid og mengdetrening.

In1000 skal være 13t jobb i uka, ikke rart at faget er vanskelig hvis man jobber mindre enn det.

Dere må gjøre oppgaver for å få til eksamen, oppgaver er alfa og omega.

Jobbe i grupper

Vi skal jobbe mye i grupper i dag, håper dere blir godt kjent! :)

Snakk sammen tre og tre

- 1) Navn, hvilken linje dere går, hvor dere er fra, funfact
- 2) Hva er dere stolte av å ha fått til i in1000?
- 3) Hvilke temaer henger dere bak på? Hva er vanskelig å forstå? Konkrete spørsmål?
- 4) Hver gruppe formulerer fire spørsmål som dere legger inn på mentimeter:
<https://www.menti.com/g9975zmxv>
menti.com, kode: 19 95 00 1
- 5) Stem på de tingene dere også lurer på!

Variabler

...

Hva er en variabel?

- Et variabelnavn som peker på en verdi
- Vi assosierer et navn med en verdi vi har lyst til å lagre
- Husk at alle variabelnavn må være i ett ord!



Eksempler

- Siden alle variabelnavn må være ett ord bruker vi i python understrek _ mellom ordene. Verdiene som er stringer kan være flere ord, "Karl" kunne godt vært "Karl Gustav".



```
min_variabel = "Karl"
```



```
mitt_tall = 10
```

Gi en variabel ny verdi

Hvis vi endrer verdien til variabelen vår flyttes merkelappen til den nye verdien.

```
min_variabel = "Karl"  
min_variabel = "Klara"
```



Typer

...

Hvilke typer har vi?

Hva slags type er verdiene våre?

- string (str): "Dette er en string"
- heltall (integer/int): $5 + 5 = 10$
- float: $5.2 + 5.5 = 10.7$
- boolean: True/False

String

```
min_variabel = "Her, mellom fnuttene, kan man skrive en tekst"  
ny_variabel = "Teksten kan ha tall: 1, 4.5, det er fremdeles en string"
```

Alle tall kan gjøres om til stringer slik med funksjonen `str()`:

```
str(5)    >>"5"
```

```
str(5.5)  >>"5.5"
```

Heltall (integer/int)

Heltall er tall uten desimal. 5.0, 5.5 og 5.14 er ikke heltall, 5 er et heltall.

Man kan gjøre om en string av et heltall til et heltall med funksjonen `int()`:

```
int("5")    >>5
```

```
int("365")  >>365
```

Float

Float, eller flyttall på norsk, er tall med desimal. Feks. 5.0, 5.5 og 5.14

Man kan gjøre om et flyttall i en string til flyttall:

```
float("5.5") >>5.5
```

```
float("6.5") >>6.5
```


Addere verdier

Man kan addere tall:

```
5 + 5      >>10
```

```
5.5 + 5    >>10.5
```

Og man kan addere stringer (det heter konkatenering):

```
"Hei " + "på deg " + "!"    >>"Hei på deg!"
```

Men man kan ikke addere dem sammen:

```
"Hei" + 5    >>ERROR
```

Boolean

Se for deg en lysbryter, enten er den av/`False` eller så er den på/`True`.

Så denne typen har bare to verdier: `False` og `True` (husk stor forbokstav)

Boolske uttrykk

...

Hva er et boolsk uttrykk?

Et boolsk uttrykk evaluerer til enten `True` eller `False`

Alle disse evaluerer til en boolean:

```
"a" == "a" >>True
```

```
1 != 2 >>True "!=" leses "er ikke lik"
```

```
1 == "1" >>False
```

```
1 < 1 >>False "<" leses "er mindre enn"
```

```
1 <= 1 >>True "<=" leses "er mindre enn eller lik"
```

Not

Dersom et uttrykk er usant, vil `not()` av uttrykket være sant, og omvendt.

```
not(False) > True
```

```
not(True) > False
```

```
not(5 == 5) > False
```

```
not(5 > 1) > False
```

```
not(5 > 20) > True
```

```
not(5 == 1) > True
```

Diskuter hva disse blir:

```
not("a" == "a") >>False
```

```
not(1 != 2) >>False
```

```
not(1 == "1") >>True
```

```
not(1 < 1) >>True
```

```
not(1 <= 1) >>False
```

Sette sammen boolske uttrykk

Vi kan sette sammen flere boolske uttrykk, og vi skal lære to måter å gjøre det på:

- 1) And
- 2) Or

And

Begge uttrykkene må være sanne, både a og b!

```
False and False > False
```

```
False and True > False
```

```
True and True > True
```

```
(5 == 1) and (5 > 20) > False
```

```
(5 == 1) and (5 > 1) > False
```

```
(5 == 5) and (5 > 1) > True
```

Or

Minst ett av uttrykkene må være sanne, enten a, eller b, eller begge!

```
False or False > False
```

```
False or True > True
```

```
True or True > True
```

```
(5 == 1) or (5 > 20) > False
```

```
(5 == 1) or (5 > 1) > True
```

```
(5 == 5) or (5 > 1) > True
```


If-tester

...

If-tester sjekker boolske uttrykk

Husk at alle boolske uttrykk evaluerer til enten `True` eller `False`! If testene sjekke boolske uttrykk og utfører koden hvis det boolske uttrykket er `True`.

```
if(True):  
    #Do this  
elif(True):  
    #Do this  
else:  
    #Do this  
  
#Man kan også ha en if-sjekk uten en  
elif-sjekk eller else-sjekk:  
  
if(True):  
    #Do this
```

Eksempler

```
if(5 < 1):  
    print("Hei på deg!")  
else:  
    print("Hade!")  
  
# "Hade!" printes alltid
```

```
if(5 == 5):  
    print("Hei på deg!")  
else:  
    print("Hade!")  
  
# "Hei på deg!" printes alltid
```

Samlinger

...

[0, 1, 5, 3, -2]

Hva er en liste?

- Som en parkeringsplass med plassnummer.
- Hver plass har plass til en bil.
- Kan dermed holde på flere verdier.
- Kan legge til verdier eller ta verdier ut av lista.



Nyttig funksjonalitet

- Lage en liste. (index) 0 1 2 3

```
parkeringsplass = ["ledig", "Volvo", "BMW", "ledig"]
```

- Indeksering - bestemme hvilken parkeringsplass vi vil se på.

```
parkeringsplass[2] # Får ut verdien på index 2 → "BMW"
```

- Vi kan også legge til ny biler i lista

```
parkeringsplass.append("Skoda")
```

Ordbok

- Som en telefonbok
 - En nøkkelverdi - analogi -> Navn i en telefonbok
 - En innholdsverdi - analogi -> Telefonnummeret som hører til navnet

- Lage en ordbok:

```
telefonbok = {
```

```
#   Nøkkel      Innholdsverdi
```

```
  "Lise"      : 41554365,
```

```
  "Ole"       : 49865732,
```

```
  "Mor"       : 47849302
```

```
}
```

Nyttig funksjonalitet

- Hente ut en verdi:
 - telefonbok[<nøkkelverdi>]
 - telefonbok[“Mor”] # → henter ut verdien 47849302
- Legge til element:
 - telefonbok[<ny nøkkel>] = <ny innholdsverdi>
 - telefonbok[“Far”] = 91432453
 - print(telefonbok[“Far”]) #Skriver nå ut 91432453
- Kan ikke ha flere like nøkkelverdier, da overskriver man kun innholdsverdien.
- len(telefonbok) # henter antall elementer i ordboken.

```
telefonbok = {
```

```
#   Nøkkel      Innholdsverdi
    “Lise”     : 41554365,
    “Ole”      : 49865732,
    “Mor”      : 47849302
```

```
}
```


Løkker



Repeterende kode

While-løkker

- Repetisjon av en kodeblokk så lenge et sannhetsuttrykk er sant.

```
while <sannhetsuttrykk/noe som evaluerer til True eller False>:  
    <repeterende kode>
```

- Eks: gjør noe så lenge bruker skriver inn et positivt tall.

```
bruker_tall = int(input("Skriv et tall: "))  
while bruker_tall > 0:  
    bruker_tall = int(input("Skriv et nytt tall: "))  
print("Du skrev ut en negativt tall og er ute av while-løkken")
```

For-løkker

2 typer for-løkker

1. for-løkke på en liste
 - går igjennom elementer i en liste
2. for-løkke med teller
 - gjør noe et gitt antall ganger med en teller
 - eks skrive ut tall fra 0 - 10

For-løkker på lister

for <variabelnavn> in liste:

 <gjør noe>

liste = [5, 3, 7, 4, 1, "Hello", -1.3]

for element in liste:

 print(element) # skriver ut ett og ett element. Live eksempel

For-løkker med teller (gjør noe x antall ganger)

```
for <variabelnavn> in range(<antall ganger>):  
    <gjør noe>
```

```
for teller in range(10):  
    print(teller) # skriver ut tallene fra 0 til 9, så printer ut 10 tall.
```

Funksjoner, prosedyrer og skop

...

Hva er en prosedyre og hvorfor er det lurt å bruke ?

- Strukturere kode i blokker
- Hvordan lage en prosedyrer?
 - a. Start med prosedyre definisjonen, du må selv gi den et navn (i eksemplet nedenfor er det minProsedyre. Du kan legge til så mange parametere som du ønsker (i eksempelet nedenfor har vi lagd to parameter1 og parameter2

```
2  
3 def minProsedyre(parameter1, parameter2):  
4
```

Hva er en prosedyre og hvorfor er det lurt å bruke ?

- Struktur kode i blokker
- Hvordan lage en prosedyrer?
 - a. Start med prosedyre definisjonen, du må selv gi den et navn (i eksemplet nedenfor er det minProsedyre. Du kan legge til så mange parametere som du ønsker (i eksempelet nedenfor har vi lagd to parameter1 og parameter2
 - b. Videre legger vi til den kode blokken vi ønsker at skal være inne i prosedyren (i dette eksempelet slår vi sammen de to parametrene, før vi legger til parameter1 en gang til bakerst og printer det ut)

```
2
3 def minProsedyre(parameter1, parameter2):
4     tekst = parameter1 + parameter2
5     tekst += parameter1
6     print(tekst)
```

Ordforklaringer:

- Parameter: variabel i prosedyren som tar i mot en verdi

Prosedyrer

- Strukturer kode i blokker
- Hvordan lage en prosedyrer?
 - a. Start med prosedyre definisjonen, du må selv gi den et navn (i eksemplet nedenfor er det minProsedyre. Du kan legge til så mange parametere som du ønsker (i eksempelet nedenfor har vi lagt to parameter1 og parameter2
 - b. Videre legger vi til den kode blokken vi ønsker at skal være inne i prosedyren (i dette eksempelet slår vi sammen de to parametrene, før vi legger til parameter1 en gang til bakerst og printer det ut)
 - c. Vi må så kalle på prosedyren, (kodeblokken inne i) prosedyren vil kjøre når den blir kalt på :) Husk å gi et argument per parameter

```
2
3 def minProsedyre(parameter1, parameter2):
4     tekst = parameter1 + parameter2
5     tekst += parameter1
6     print(tekst)
7
8 minProsedyre("argument1", "argument2")
```

Ordforklaringer:

- Parameter: variabel i prosedyren som tar i mot en verdi
- Argument: verdi sendt inn når prosedyren kalles

Fordeler med prosedyrer

- Hjelper oss å strukturere koden
- Hvis vi har relativ lik kode flere steder burde dette lages til en prosedyre slik at vi kan kalle på den, istedenfor å ha duplikater i koden

Parameteroverføring på 1,2,3

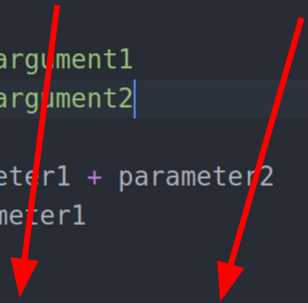
- Fordeler med parameter er at vi kan ha tilpasninger i prosedyren

```
3 def minProsedyre(parameter1, parameter2):
4     """
5     parameter1 = argument1
6     parameter2 = argument2
7     """
8     tekst = parameter1 + parameter2
9     tekst += parameter1
10    print(tekst)
11
12 minProsedyre("argument1", "argument2")
```

Parameteroverføring på 1,2,3


- Fordeler med parameter er at vi kan ha tilpasninger i prosedyren

```
3 def minProsedyre(parameter1, parameter2):  
4     """  
5     parameter1 = argument1  
6     parameter2 = argument2  
7     """  
8     tekst = parameter1 + parameter2  
9     tekst += parameter1  
10    print(tekst)  
11  
12 minProsedyre("argument1", "argument2")
```

Two red arrows originate from the function signature in the code block. One arrow points from the parameter 'parameter1' on line 3 down to the argument 'argument1' on line 12. The other arrow points from the parameter 'parameter2' on line 3 down to the argument 'argument2' on line 12.

Kodeflyt

```
3 def minProsedyre(parameter1, parameter2):
4     """
5     parameter1 = argument1
6     parameter2 = argument2
7     """
8     tekst = parameter1 + parameter2
9     tekst += parameter1
10    print(tekst)
11
12    minProsedyre("argument1", "argument2")
```



Kodeflyt

```
3 def minProsedyre(parameter1, parameter2):  
4     """  
5     parameter1 = argument1  
6     parameter2 = argument2  
7     """  
8     tekst = parameter1 + parameter2  
9     tekst += parameter1  
10    print(tekst)  
11  
12    minProsedyre("argument1", "argument2")
```

Variabler i minProsedyre

parameter1	“argument1”
parameter2	“argument2”

Kodeflyt

```
3 def minProsedyre(parameter1, parameter2):
4     """
5     parameter1 = argument1
6     parameter2 = argument2
7     """
8     tekst = parameter1 + parameter2
9     tekst += parameter1
10    print(tekst)
11
12 minProsedyre("argument1", "argument2")
```

Variabler i minProsedyre

parameter1	"argument1"
parameter2	"argument2"
tekst	"argument1argument2"

Kodeflyt

```
3 def minProsedyre(parameter1, parameter2):
4     """
5     parameter1 = argument1
6     parameter2 = argument2
7     """
8     tekst = parameter1 + parameter2
9     tekst += parameter1
10    print(tekst)
11
12 minProsedyre("argument1", "argument2")
```



Variabler i minProsedyre

parameter1	"argument1"
parameter2	"argument2"
tekst	"argument1argument2argument1"

Kodeflyt

```
3 def minProsedyre(parameter1, parameter2):
4     """
5     parameter1 = argument1
6     parameter2 = argument2
7     """
8     tekst = parameter1 + parameter2
9     tekst += parameter1
10    print(tekst)
11
12 minProsedyre("argument1", "argument2")
```



Variabler i minProsedyre

parameter1	"argument1"
parameter2	"argument2"
tekst	"argument1argument2argument1"

På terminalen:

"argument1argument2argument1"

Hva er en funksjon ?

- En funksjon lar deg “outsource” en beregning til en separat kodeblokk
- Forskjellen på en funksjon og en prosedyre er altså at en funksjon returnerer noe det gjør ikke en prosedyre
- Defineres på samme måte som en prosedyre bortsett fra at at vi må ha return med

```
2
3 def storstTall(tall1, tall2):
4     if tall1 > tall2
5         return tall1
6     return tall2
```

Kodeflyt

```
3 def storstTall(tall1, tall2):
4     if tall1 > tall2
5         return tall1
6     return tall2
7
8
9 storstEn = storstTall(1, 2)
10
11 print("Det største tallet av 1 og 2 er ", storstEn)
12
13 storstTo = storstTall(6, 2)
14
15 print("Det største tallet av 6 og 2 er ", storstTo)
```



Kodeflyt

```
def storstTall(tall1, tall2):  
4     if tall1 > tall2  
5         return tall1  
6     return tall2  
7  
8 storstEn = storstTall(1, 2)  
9 print("Det største tallet av 1 og 2 er ", storstEn)  
10  
11 storstTo = storstTall(6, 2)  
12 print("Det største tallet av 6 og 2 er ", storstTo)
```

Variabler i storstTall

tall1	1
tall2	2

Kodeflyt

```
3 def storstTall(tall1, tall2):  
4     if tall1 > tall2  
5         return tall1  
6     return tall2  
7  
8 storstEn = storstTall(1, 2)  
9 print("Det største tallet av 1 og 2 er ", storstEn)  
10  
11 storstTo = storstTall(6, 2)  
12 print("Det største tallet av 6 og 2 er ", storstTo)
```

Variabler i storstTall

tall1	1
tall2	2

Kodeflyt

```
3 def storstTall(tall1, tall2):
4     if tall1 > tall2
5         return tall1
6     return tall2
7
8 storstEn = storstTall(1, 2)
9 print("Det største tallet av 1 og 2 er ", storstEn)
10
11 storstTo = storstTall(6, 2)
12 print("Det største tallet av 6 og 2 er ", storstTo)
```

Variabler i storstTall

tall1	1
tall2	2

Kodeflyt

```
3 def storstTall(tall1, tall2):
4     if tall1 > tall2
5         return tall1
6     return tall2
7
8 storstEn = storstTall(1, 2) 2
9 print("Det største tallet av 1 og 2 er ", storstEn)
10
11 storstTo = storstTall(6, 2)
12 print("Det største tallet av 6 og 2 er ", storstTo)
```

Variabler i det globale skop

storstEn	2
----------	---

Kodeflyt

```
3 def storstTall(tall1, tall2):
4     if tall1 > tall2
5         return tall1
6     return tall2
7
8 storstEn = storstTall(1, 2) 2
9 print("Det største tallet av 1 og 2 er ", storstEn)
10
11 storstTo = storstTall(6, 2)
12 print("Det største tallet av 6 og 2 er ", storstTo)
```

På terminalen:

Det største tallet av 1 og 2 er 2

Variabler i det globale skop

storstEn	2
----------	---

Kodeflyt

```
3 def storstTall(tall1, tall2):
4     if tall1 > tall2
5         return tall1
6     return tall2
7
8 storstEn = storstTall(1, 2) 2
9 print("Det største tallet av 1 og 2 er ", storstEn)
10
11 → storstTo = storstTall(6, 2)
12 print("Det største tallet av 6 og 2 er ", storstTo)
```

På terminalen:

Det største tallet av 1 og 2 er 2

Variabler i det globale skop

storstEn	2
----------	---

Kodeflyt

```
def storstTall(tall1, tall2):  
    4     if tall1 > tall2  
    5         return tall1  
    6     return tall2  
    7  
    8     storstEn = storstTall(1, 2) 2  
    9     print("Det største tallet av 1 og 2 er ", storstEn)  
    10  
    11     storstTo = storstTall(6, 2)  
    12     print("Det største tallet av 6 og 2 er ", storstTo)
```

På terminalen:

Det største tallet av 1 og 2 er 2

Variabler i storstTall

tall1	6
tall2	2

Variabler i det globale skop

storstEn	2
----------	---

Kodeflyt

```
3 def storstTall(tall1, tall2):  
4     if tall1 > tall2  
5         return tall1  
6     return tall2  
7  
8 storstEn = storstTall(1, 2) 2  
9 print("Det største tallet av 1 og 2 er ", storstEn)  
10  
11 storstTo = storstTall(6, 2)  
12 print("Det største tallet av 6 og 2 er ", storstTo)
```

På terminalen:

Det største tallet av 1 og 2 er 2

Variabler i storstTall

tall1	6
tall2	2

Variabler i det globale skop

storstEn	2
----------	---

Kodeflyt

```
3 def storstTall(tall1, tall2):
4     if tall1 > tall2
5         return tall1
6     return tall2
7
8 storstEn = storstTall(1, 2) 2
9 print("Det største tallet av 1 og 2 er ", storstEn)
10
11 storstTo = storstTall(6, 2)
12 print("Det største tallet av 6 og 2 er ", storstTo)
```

På terminalen:

Det største tallet av 1 og 2 er 2

Variabler i storstTall

tall1	6
tall2	2

Variabler i det globale skop

storstEn	2
----------	---

Kodeflyt

```
3 def storstTall(tall1, tall2):
4     if tall1 > tall2
5         return tall1
6     return tall2
7
8 storstEn = storstTall(1, 2) 2
9 print("Det største tallet av 1 og 2 er ", storstEn)
10
11 storstTo = storstTall(6, 2) 6
12 print("Det største tallet av 6 og 2 er ", storstTo)
```

På terminalen:

Det største tallet av 1 og 2 er 2

Det største tallet av 6 og 2 er 6

Variabler i det globale skop

storstEn	2
storstTo	6

Skop

Kort sagt har alle tilgang på variablene som ligger i det globale skopet, mens det er en variabel inne i en prosedyre/funksjon er det kunne denne prosedyren/funksjonen, som har tilgang og kan bruke denne variabelen

Skop

```
2
3 def prosedyre():
4     a = 10
5     b = 23
6     print(a + b)
7
8 def funksjon():
9     c = 34
10    d = 19
11    return c + d
12
13 k = "hei du"
14 l = 3.9
```

Skop

prosedyre kan bruke
variablene: a,b,k,l

funksjon kan bruke
variablene: c,d,k,l

det globale skopet kan
kun bruke variablene: k og
l

```
2
3 def prosedyre():
4     a = 10
5     b = 23
6     print(a + b)
7
8 def funksjon():
9     c = 34
10    d = 19
11    return c + d
12
13 k = "hei du"
14 l = 3.9
```

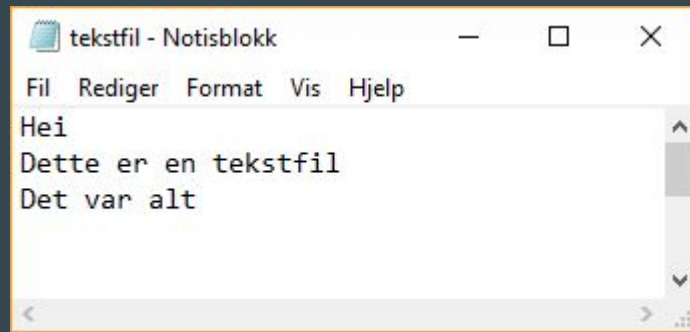

Filer og filinnlesing



Kort tema, store bruksområder

Filer inneholder informasjon

Må “åpnes” for at programmet skal bruke den



```
innfil = open("tekstfil.txt")
```

innfil-variabelen inneholder nå et fil-objekt!

Dette fil-objektet skal vi bruke for å hente ut filens innhold

Forskjellige typer fil-objekter

Read-filobjekter: Kan leses av i programmet

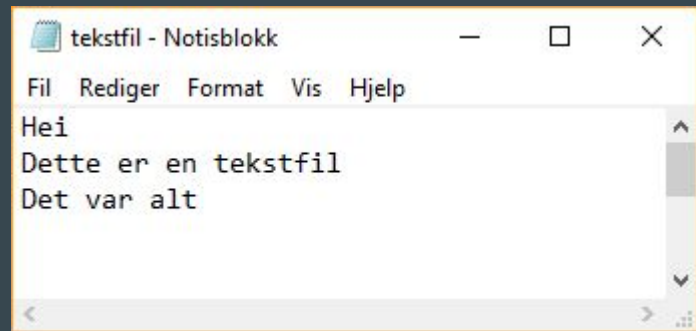
Write-filobjekter: Sletter innholdet i en fil, og skriver nytt innhold

Append-filobjekter: skriver nytt innhold på slutten av en fil

Read-filobjekter

Har parameteret **“r”**, som lar oss leses av

Har (ofte) parameteret **encoding = “utf-8”** som lar deg lese spesielle tegn, som æ, ø, å

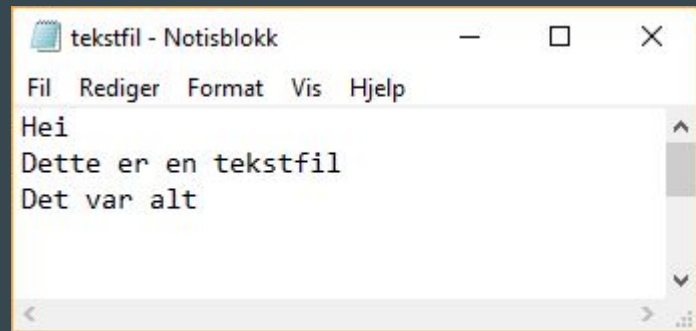


```
innfil = open("tekstfil.txt", "r", encoding = "utf-8")
```

```
linje = innfil.readline() -----> linje = "hei\n"
```

```
linje = linje.strip() -----> linje = "hei"
```

Kan lese flere linjer



```
innfil = open("tekstfil.txt", "r")
```

```
linje1 = innfil.readline() -----> linje1 = "hei\n"
```

```
linje2 = innfil.readline() -----> linje2 = "Dette er en tekstfil\n"
```

```
linje3 = innfil.readline() -----> linje3 = "Det var alt\n"
```

Kan inkludere `strip()` for å slippe newline på slutten

```
linje = innfil.readline().strip() -----> linje = "linje i fil, naa uten en newline"
```

For-løkker på filobjekter

```
innfil = open("tekstfil.txt", "r")
```

```
for linje in innfil:
```

```
    print(linje)
```

```
for linje in innfil:
```

```
    print(linje.strip())
```

```
Hei
```

```
Dette er en tekstfil
```

```
Det var alt
```

```
Hei
```

```
Dette er en tekstfil
```

```
Det var alt
```

Dele opp linjer i fil

```
innfil = open("arbeid.txt", "r")
```

```
for linje in innfil:
```

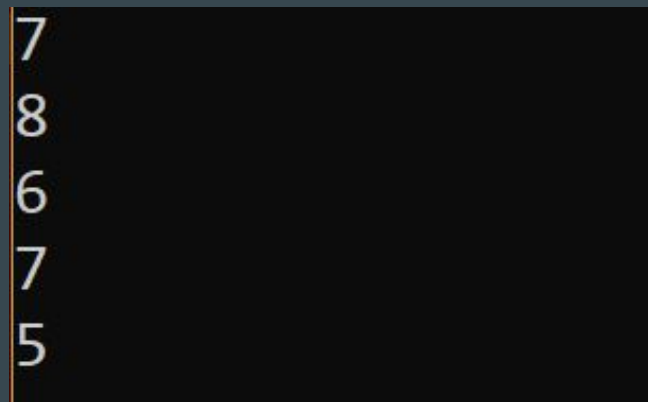
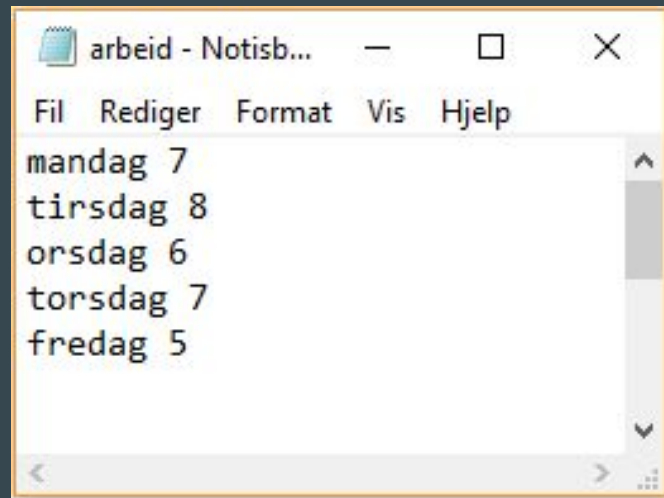
```
    linje = linje.strip()
```

```
    biter = linje.split(" ")
```

```
    print(biter[1])
```

Husk: alle linjer i filer er tekststrenger!

Cast verdier til noe annet hvis det trengs



Skrive til fil

```
utfil = open("filnavn.txt", "w")
```

```
utfil = open("filnavn.txt", "a")
```

write overskriver en fil, append legger til slutten av en fil

Hvis ikke filnavnet eksisterer, vil det lages en ny fil med filnavnet

```
utfil.write("setning som skal skrives til fil\n")
```