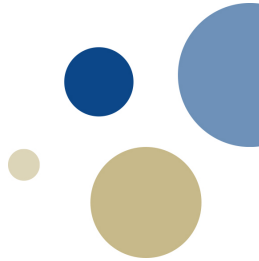
Funksjoner



Svært ofte ønsker man å gjenbruke deler av koden man skriver.

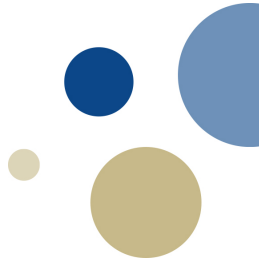
Funksjoner

Svært ofte ønsker man å gjenbruke deler av koden man skriver. Det å printe / skrive til skjermen er egentlig en veldig kompleks prosess, men den har blitt *abstrahert bort*.
Abstraksjoner er veldig viktige for å løse oppgaver, både til eksamen og senere.
Dette begrepet kommer vi tilbake til når vi begynner å se på eksamensoppgaver.

Funksjoner

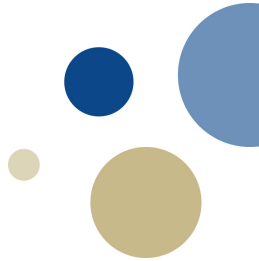
Et viktig prinsipp er det å *kalle* en funksjon. Når man *definerer* en funksjon, lager man bare en generell oppskrift for ubestemte parametere.
Når man kaller på en funksjon, så utfører man den oppskriften. Det vil si at vi gir det noen *fysiske* verdier å jobbe med.

Funksjoner - Funksjoner uten retur-verdi



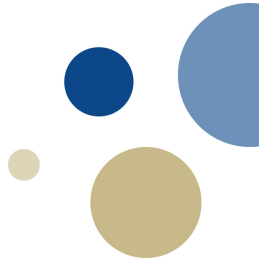
I mange tilfeller skal en funksjon gjøre noe enkelt som ikke krever at koden husker på endringen etter funksjonen har blitt kjørt.

Funksjoner - Funksjoner uten retur-verdi



I mange tilfeller skal en funksjon gjøre noe enkelt som ikke krever at koden husker på endringen etter funksjonen har blitt kjørt.
Dette er ofte relatert til *output*, som f.eks. printing og skrive til fil.

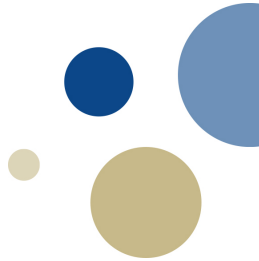
Funksjoner - Funksjoner uten retur-verdi



Funksjoner begynner med kodeordet *def*, funksjonsnavnet, en parentes med alle parameterene funksjonen tar inn og **VIKTIG** kolon :. Det må også være minst en kodesnutt i funksjonen.
Kanskje den enkleste funksjonen?

```
def func():
    pass
```

Funksjoner - Funksjoner uten retur-verdi



Eksempel med printing.

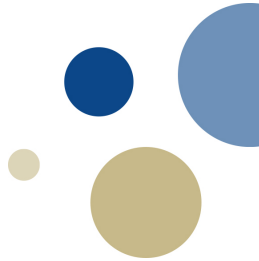
```
def print_resultater(res, tekst):
    print("Resultatet ble",res,sep=": ",end=" og ")
    print("dette er teksten: " + tekst)
```

Funksjoner - Funksjoner uten retur-verdi

Vi har allerede jobbet med default-parametere. *sep* og *end* er default-verdier i *print*-funksjonen, og med mindre de spesifiseres vil *sep* alltid være " " (mellomrom) og *end* alltid være "\n" (ny linje). Her bruker vi det til å bestemme om vi skal printe eller lagre til fil.

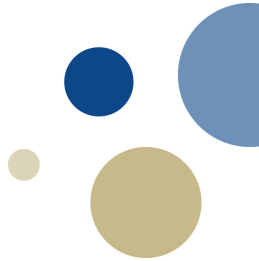
```
def output_resultater(res, file_path=""):
    if file_path != "":
        f = open(file_path, "w")
        f.write(res)
        f.close()
    else:
        print("Her er resultatet:", res)

output_resultater("hei") # printes
output_resultater("hei", "test.txt") # skrives til fil
output_resultater("hei", file_path="test.txt")
```

Funksjoner - Funksjoner med retur-verdi



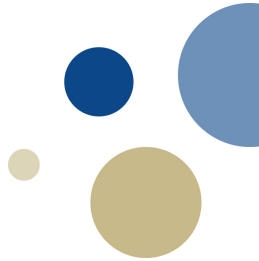
Dersom man ikke returnerer en verdi fra en funksjon, kan man ikke egentlig forvente at noe endres utenfor funksjonen vår.

Funksjoner - Funksjoner med retur-verdi



Dersom man ikke returnerer en verdi fra en funksjon, kan man ikke egentlig forvente at noe endres utenfor funksjonen vår.
Dette er ikke helt korrekt, men kommer tilbake til dette senere.

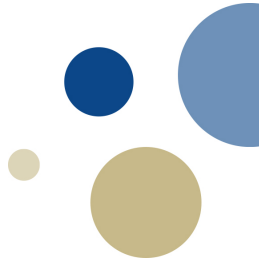
Funksjoner - Funksjoner med retur-verdi



Det vi vil skal komme ut av funksjonen vår, det som skal returneres, skriver vi etter return.

```
def pluss(tall_1, tall_2):
    tall_sum = tall_1 + tall_2
    return tall_sum
```

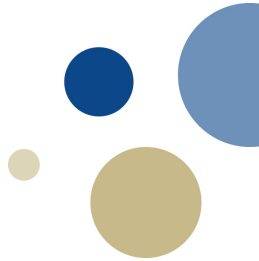
Funksjoner - Funksjoner med retur-verdi



Viktig: Når en verdi returneres fra en funksjon, så kan det tolkes som at vi *bytter* ut funksjonskallet med det som beregnes i og returneres av funksjonen.

```
def pluss(tall_1, tall_2):
    tall_sum = tall_1 + tall_2
    return tall_sum

resultat = pluss(1, 2)
print(resultat)
```

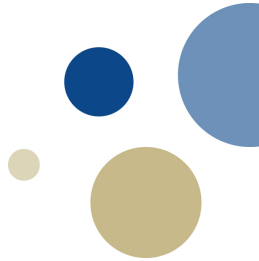
Funksjoner - Funksjoner med retur-verdi



Viktig: Når en verdi returneres, må den tilordnes, ellers glemmer programmet hva resultatet er.
I noen tilfeller trenger vi kanskje ikke resultatet til noe annet enn f.eks. printing, men ofte er det lurt å tilordne.

```
def pluss(tall_1, tall_2):
    tall_sum = tall_1 + tall_2
    return tall_sum

resultat = pluss(1, 2)
print(resultat)
```

Funksjoner - Funksjoner med retur-verdi



Styrken med funksjoner er at de kan være ganske generelle, og spesifiseres kun når de kalles.
Sammen med løkker og lister er funksjoner noe av det viktigste med programmering.