

Outline

- Lese, bruke og skrive data i filer
- Eksempel: kombinere data fra ulike filer
- **Mer om funksjoner: parametre og skop for variabler**

Prosedyrer versus funksjoner i Python

- Det er i Python ikke noe teknisk skille mellom prosedyre og funksjon
 - Begge kalles i Python "funksjoner" og har samme form:
`def min_funksjon(parameter):`
...
- Forskjellen er i vårt hode (vårt formål):
 - Det som skiller en "ekte" funksjon fra en prosedyre er at funksjoner returnerer en verdi man er interessert i
- Dette ser man som en **return** i koden, men også ved at resultatet av kallet brukes/tas vare på:
 - `toDusin = gang_med_to(dusin)`

Med/uten parametre/returverdi

- (En litt filosofisk distinksjon...)
- **Prosedyrer uten parametre:**
 - Handler stort sett om kontrollflyt
- **Prosedyrer med parametre:**
 - Handler stort sett om tilpasset gjenbruk av kode
- **Funksjoner:**
 - Outsourcer en overordnet beregning (transformerer inn til ut)
 - Inneholder vanligvis ikke input og print (!)

Tre kilder til funksjoner

- Innebygde funksjoner
 - `print`, `int`, `input` ...
 - Følger med Python og er alltid tilgjengelige
- Funksjoner i standard-biblioteket (ikke innebygde)
 - `sqrt`, `ctime`
 - Er del av en modul: `math`, `time`, ...
 - ```
from random import randint
terning = randint(1,6)
```
- Egendefinerte funksjoner
  - `def min_funksjon(parameter): ...`

# *Fra uke4:* Prosedyre med parametre

```
def mittProsedyreNavn(parameter1, parameter2, ...):
 kodelinje1
 kodelinje2
 ...
```

For å kjøre alle kodelinjene i prosedyren ("*kalle*  
prosedyren"):

```
mittProsedyreNavn(argument1, argument2, ...)
```

# Parameteren tilordnes verdien av argumentet!

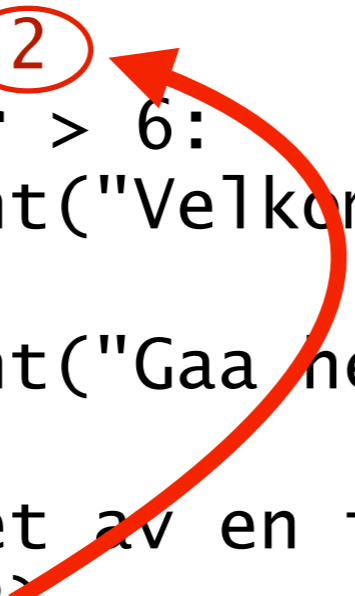
```
def skrivAlder(alder):
 if alder > 6:
 print("Velkommen til mitt program");
 else:
 print("Gaa heller ut og lek i skogen");

print("Hacket av en toaaring: ")
skrivAlder(2)
```

# Parameteren tilordnes verdien av argumentet!

```
def skrivAlder(alder):
 alder = 2
 if alder > 6:
 print("Velkommen til mitt program");
 else:
 print("Gaa heller ut og lek i skogen");

print("Hacket av en toaaring: ")
skrivAlder(2)
```



# *Fra uke4:* Subrutiner med returverdi - **Funksjoner**

```
def mittFunksjonsNavn(parameter1, ...):
 kode1linje1
 kode1linje2
 return beregnet_verdi
```

For å kjøre alle kodelinjene i funksjonen ("*kalle prosedyren*"):

```
verdien_jeg_trenger = mittFunksjonsNavn(argument1, ..)
```



# Funksjonskallet *evaluerer til returverdien*


```
def gi_meg_pi():
 return 3.14
```

```
pi = gi_meg_pi()
print(pi)
```

# Funksjonskallet *evaluerer til returverdien*

```
def gi_meg_pi():
 return 3.14
```

```
 3.14
pi = gi_meg_pi()
print(pi)
```



# Funksjonskallet *evaluerer til returverdien*

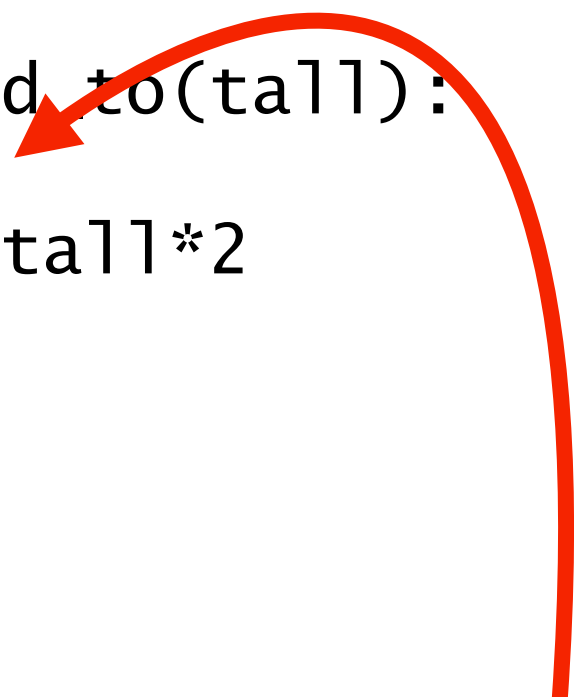
```
def gang_med_to(tall):
 return tall*2
```

```
dusin=13
toDusin = gang_med_to(dusin)
print(toDusin)
```

# Funksjonskallet *evaluerer til returverdien*

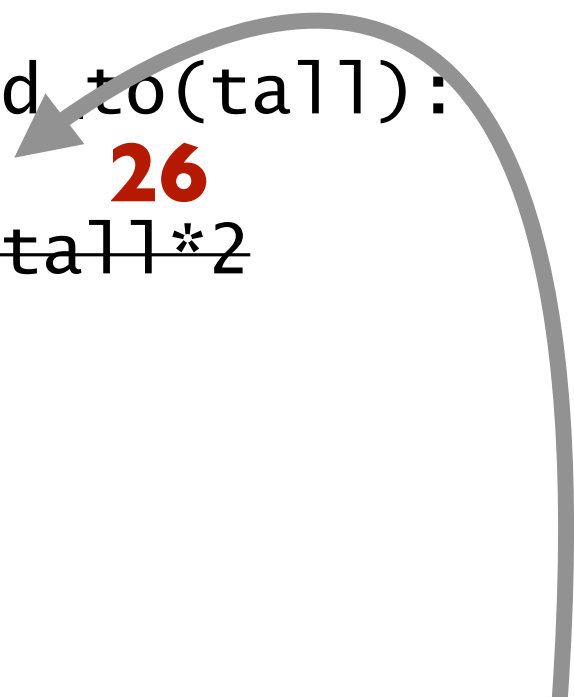
```
def gang_med_to(tall):
 tall=13
 return tall*2
```

```
dusin=13
toDusin = gang_med_to(dusin)
print(toDusin)
```



# Funksjonskallet *evaluerer til returverdien*

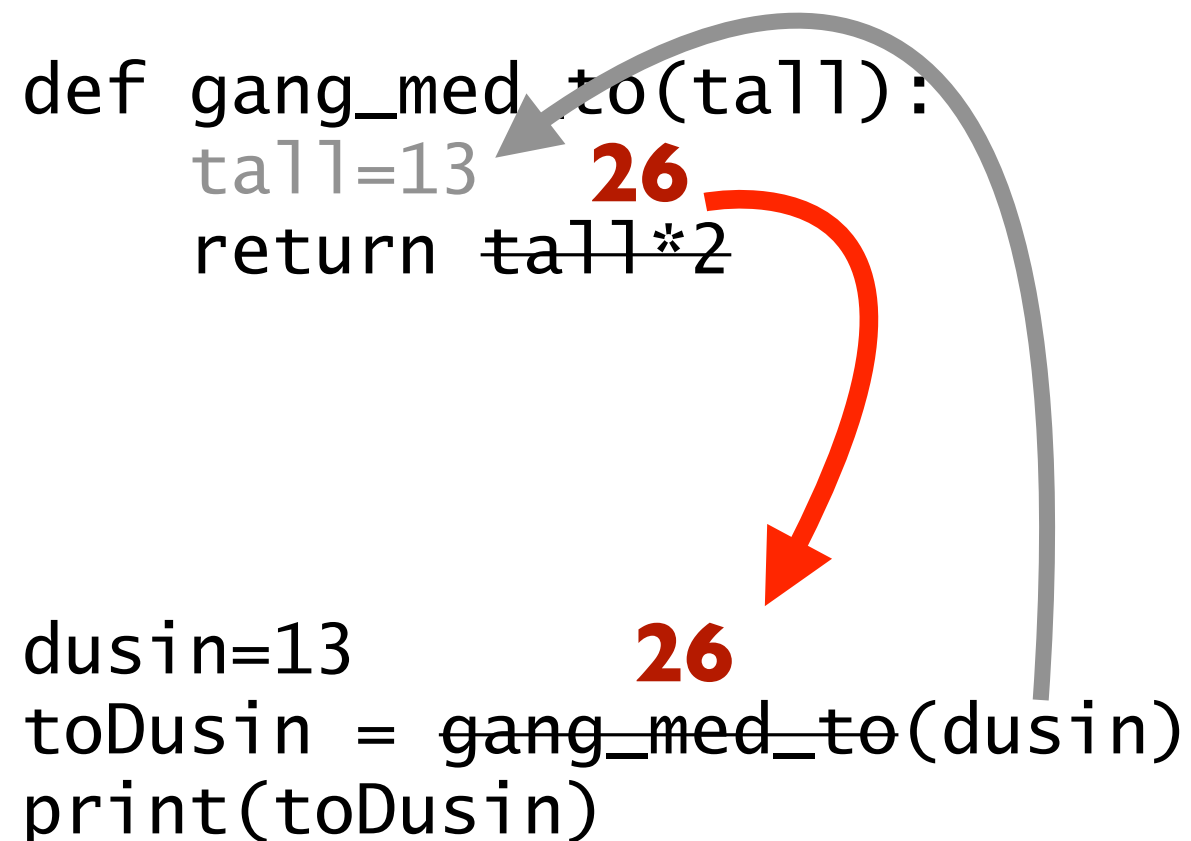
```
def gang_med_to(tall):
 tall=13 26
 return tall*2
```



```
duzin=13
toDusin = gang_med_to(duzin)
print(toDusin)
```

# Funksjonskallet *evaluerer* til *returverdien*

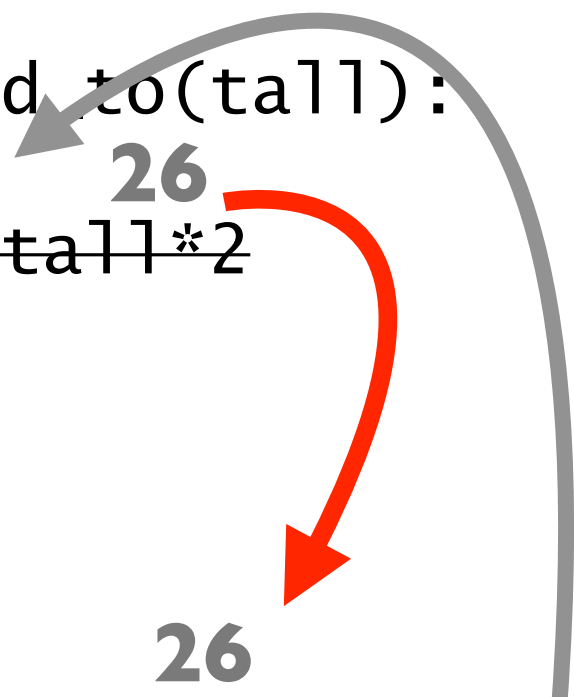
```
def gang_med_to(tall):
 tall=13
 return tall*2
```



```
dusin=13
toDusin = gang_med_to(dusin)
print(toDusin)
```

# Funksjonskallet *evaluerer til returverdien*

```
def gang_med_to(tall):
 tall=13
 return tall*2
```



```
dusin=13
toDusin = gang_med_to(dusin)
print(toDusin)
26
```

# Skop for variable

- Hver funksjon har sine egne lokale variabler
  - Variablene som tilhører én funksjon er ikke tilgjengelige i en annen funksjon
  - **Skopet** til en variabel er hvor i koden denne variabelen er tilgjengelig - hvilken funksjon den tilhører
  - Enhver variabel tilhører et *navnerom*, som igjen er tilknyttet en *ramme* - for vårt formål er dette som synonymmer for skop
- Vi overfører verdier mellom funksjoner gjennom parametre og returverdier
  - Vi kan sende inn argument til en funksjon vi kaller
  - Vi kan få en returverdi tilbake fra en funksjon vi kaller
- [velkomst.py]



# Det "globale" skopet

- Variabler som ikke er del av en funksjon kalles globale
  - Disse globale variablene er spesielle, fordi de i prinsippet er tilgjengelige inni alle funksjoner i filen
- I IN1000 unngår vi globale variable!
  - Et viktig poeng med funksjoner er at man lett ser hva som kommer inn (argumenter) og ut (returverdier)
- Siden globale variabler i prinsippet er tilgjengelige, kan de bli brukt ved et "uhell" ([velkomst2.py])
  - Slike uhell kan føre til fremtidige bugs og feilsøking
  - Dette kan unngås ved å flytte kode fra ytterste nivå til en prosedyre "hovedprogram" ([velkomst3-4])

# Oppgave:

*hva skrives ut her?*

```
def funksjon():
 tekst = "Jeg elsker Oslo!"
 print(tekst)
```

```
tekst = "Jeg elsker Bergen!"
funksjon()
print(tekst)
```

# Svar:

*Det samme navnet representerer to helt ulike variable i det lokale og globale skopet*

## *Utskrift:*

```
Jeg elsker Oslo!
Jeg elsker Bergen!
```

```
def funksjon():
 tekst = "Jeg elsker Oslo!"
 print(tekst)
```

```
tekst = "Jeg elsker Bergen!"
funksjon()
print(tekst)
```

# Svar:

*Siden det uansett er to helt ulike variable:  
ingen forskjell om variablene hadde ulike navn*

## *Utskrift:*

```
Jeg elsker Oslo!
Jeg elsker Bergen!
```

```
def funksjon():
 tekst = "Jeg elsker Oslo!"
 print(tekst)
```

```
utsagn = "Jeg elsker Bergen!"
funksjon()
print(utsagn)
```

# Men:

*Dersom det ikke var deklarerert en lokal variabel tekst, ville dette navnet representert den globale variabelen!*

## *Utskrift:*

```
Jeg elsker Bergen!
Jeg elsker Bergen!
```

```
def funksjon():
```

```
 print(tekst)
```

```
tekst = "Jeg elsker Bergen!"
```

```
funksjon()
```

```
print(tekst)
```

# Med vår anbefaling:

*Dersom den ytre koden også flyttes til en funksjon (hovedprogram), slipper man mulig forvirring rundt det lokale og globale skopet*

## *Kjøring:*

```
NameError: global name 'tekst' is not defined
```

```
def funksjon():
 print(tekst)

def hovedprogram():
 tekst = "Jeg elsker Bergen!"
 funksjon()
 print(tekst)

hovedprogram()
```

# Med vår anbefaling:

*Om en verdi som er deklarerert i én funksjon skal brukes i en annen, må man på tydelig vis sende over som parameter*

## *Utskrift:*

```
Jeg elsker Bergen!
Jeg elsker Bergen!
```

```
def funksjon(tekst):
 print(tekst)

def hovedprogram():
 utsagn = "Jeg elsker Bergen!"
 funksjon(utsagn)
 print(utsagn)

hovedprogram()
```

# Med vår anbefaling:

*Bruk gjerne ulike navn i ulike funksjoner, men det gjør uansett ingen forskjell siden alle variable er lokale for hver sin funksjon*

## *Utskrift:*

```
Jeg elsker Bergen!
Jeg elsker Bergen!
```

```
def funksjon(tekst):
 print(tekst)

def hovedprogram():
 tekst = "Jeg elsker Bergen!"
 funksjon(tekst)
 print(tekst)

hovedprogram()
```



# Introdusere funksjoner i programmet om restplass

- Å lage nedbørs-ordbok fra data i fil er én oppgave
  - Vi definerer en funksjon som basert på et filnavn lager og returnerer en ordbok [restplass3.py]
- Den resterende analysen kan legges i en prosedyre
  - Basert på ordbok for regn og filnavn for søkeantall skriver den ut gjennomsnittlig søk på gode og dårlige dager
- Kallene til funksjonen og prosedyren over legges i en funksjon "hovedprogram"
  - Tydeliggjør hva som kjøres og sikrer at alle verdier overføres med parametre og returverdier [restplass4.py]

# Utsett til i morgen, det du ikke trenger gjøre i dag

- Funksjoner tillater å utsette problemer!
  - Fokuser først på hva som trengs overordnet
  - Deretter gå løs på detaljene
- Eksempel:

```
kvm=60
postnr=0316
inntekt=503800
pris = regn_bolig_pris(kvm, postnr)
maks_laan = regn_kreditt(inntekt)
if (maks_laan > pris):
 print("Yes!")
Deretter skriv selve funksjonene..
```