Løse reelle problemer

Løse problemer med data fra fil, samt litt mer om funksjoner

IN1000, uke 5/6
Geir Kjetil Sandve

Mål for uken

- Få enda mer trening i hvordan bruke løkker, samlinger og beslutninger for å løse problemer
  - Som del av dette skal vi lære å behandle data fra filer
- Få enda mer presis forståelse av funksjoner, spesielt parameteroverføring og returverdier
  - Som del av dette skal vi lære om skop for variabler

Outline

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

Outline

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

Innlesing fra fil

- Å hente data fra filer er gøy!
  - Man kan jobbe på mye større og mer spennende data enn fra tastatur
  - Man slipper å taste det inn hver gang man kjører
- Innlesing fra tekstfiler er veldig rett frem i Python
  - En tekstfil åpnet i Python er faktisk en *samling* av linjer
  - Man kan dermed iterere gjennom linjer i filen vha for-løkke
  - (Det finnes også mange andre måter å gjøre det på)

Hvordan lese inn fra fil

- Først åpne en fil *(biblioteksfunksjonen **open** lager et fil-objekt)*:
  - `min_fil = open("mittFilNavn.txt")`
- Deretter iterere gjennom hver linje i filen:
  - ```
    min_fil = open("mittFilNavn.txt")
    for linje in min_fil:
    ```
- Inni for-løkken kan man gjøre noe med linjen
  - ```
    min_fil = open("mittFilNavn.txt")
    for linje in min_fil:
        print("Her fant jeg: " + linje)
    ```
- [les_fra_fil.py]

Organisere informasjon innad på linjer

- Det er ofte nyttig å organisere informasjon på bestemte måter i en fil:
  - Kan gruppere: én gruppe på hver linje, og alt tilhørende gruppen inndelt langs linjen
  - Representere en tabell: hver linje er en rad, og hver linje er videre inndelt i verdier per kolonne (som i et regneark)
- Slike filer er ofte i *tabulært format*, hvor man bruker et spesifikt tegn (f.eks. komma) for å skille verdier innad i linje

Lese tabulære filer i python

- En tabulær fil er en helt vanlig tekstfil, hvor man kan iterere gjennom linjer
  - `for linje in open("min_tabular_fil.csv"):`
- Hver linje kan splittes til en liste *(basert på separator-tegn)*
  - `biter = linje.split(",")` *#Splitter på hvert komma*
  - `biter = linje.split()` *#Splitter på blanke tegn*
- Man kan hente ut bestemte biter (kolonner):
  - ```
    navn = biter[0]
    alder = int(biter[1])
    ```
- Man kan også iterere gjennom bitene:
  - ```
    for bit in biter:
        print("Jeg fant: " + bit)
    ```

Bygge samling fra innhold i fil

- Vi har tidligere laget to versjoner av hovedstad-program:
  - I uke 1 sjekket vi land med if-setninger og printet hovedstad
  - I uke 3 la vi land og hovedsted inn i en ordbok og slo opp land
- Nå vil vi bygge ordboken fra innholdet i en fil
  - Vi trenger informasjon som kobler étt land med én hovedstad
  - Bruker tabulær fil hvor hver linje har land og tilhørende hovedstad
  - Går gjennom hver linje i fil, splitter, og legger inn i ordbok
- [hovedstad.py]

Oppsummere data gruppert på linjer

- Vi vil finne hvilken måned hadde høyest dagsnedbør
  - Har tabulær fil med nebørsmengde per dag gjennom året
  - Dagsnedbør er gruppert på én linje for hver måned
- Fremgangsmåte: nøstet løkke inkludert splitting
  - Itererer gjennom måneder (linjer)
  - Splitter hver linje til liste over dagsnedbør
  - Itererer gjennom dagsnedbør og finner høyeste
  - Skriver ut høyeste verdi for denne måneden
- [regn.py]

Oppgave

- Gitt en tekstfil av typen vist under (navn og alder), skriv et program som leser filen og skriver ut navnet på den eldste personen i fila

alder.csv

odlaug:76
oluf:65
gunda:74
malfrid:80
godtfred:68

En mulig løsning

- [alder.py]

Hvordan skrive til tekstfil

- Først åpne en fil for skriving:
  - `min_fil = open("mittUtFilNavn.txt", "w")`
- Deretter skrive tekst (en streng-verdi) til slutten av filen:
  - `min_fil = open("mittUtFilNavn.txt", "w")`
    `min_fil.write("Dette havner i filen")`
- Man vil typisk ha linjeskift i det man skriver ut:
  - **`min_fil.write("Dette havner i filen\n")`**
    `min_fil.write("Dette havner på\n ulike\n linjer\n")`
- Når man er ferdig bør man lukke filen:
  - `min_fil.close()`
- [elefanter.py]

Outline

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

Eksempel: drømmer vi oss bort når det regner?

- Problemstilling:
  - Er det slik at nordmenn gjør flere internettsøk etter restplass på dager hvor det regner?
- Fremgangsmåte:
  - Lese nedbørsdata fra fil (hentet fra yr.no)
  - Lese søkestatistikk fra fil (hentet fra Google Trends)
  - Se om antall søk på "restplass" er høyere på dagene det var regn (mer enn 2mm)

Koble data

- Vi vil telle antall søk på dager det er regn
  - Vi kan skrive noe a la:
    ```
    if regn_denne_dagen:
        sum_sok_regndager += sok_denne_dagen
    ```
- Vi må da koble nedbør og søkeantall for samme dato (vi kan iterere gjennom datoer, men må få til at vi for en gitt dato samtidig får tak i *regn_denne_dagen* og *sok_denne_dagen)*
  - Informasjon om nedbør og søk for en bestemt dag ligger imidlertid i to separate filer
- Hvordan få tak i regn og søk for en gitt dag samtidig?
  - *(f.eks. skrive ut regn og søk på én linje for hver dag)*
  - Er det noe fra tidligere ukers pensum som kan hjelpe? Tenk først selv, deretter diskuter med sideperson!

Koble data

- En ordbok lar oss enkelt gå fra en dato til en verdi
- Vi kan gå gjennom nedbørsfilen og bygge en ordbok som kobler dato til nedbør (og gjøre samme for søk)
- Etterpå itererer vi gjennom datoer
  - Når vi behandler en gitt dato slår vi opp korresponderende søk og nedbør i ordbøkene
- [restplass1.py]

Gjøre selve analysen

- Når vi har fått tak i søk og nedbør for samme dag
  - I stedet for å skrive ut finner vi gjennomsnittlig søkescore for gode dager og for dårlige dager
  - Vi teller antall og summerer søkescore for gode dager (og gjør det samme for dårlige)
  - Til slutt regner vi ut gjennomsnitt (sum/antall)
- [restplass2.py]

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, ...):
    kodelinje1
    kodelinje2
    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)
```

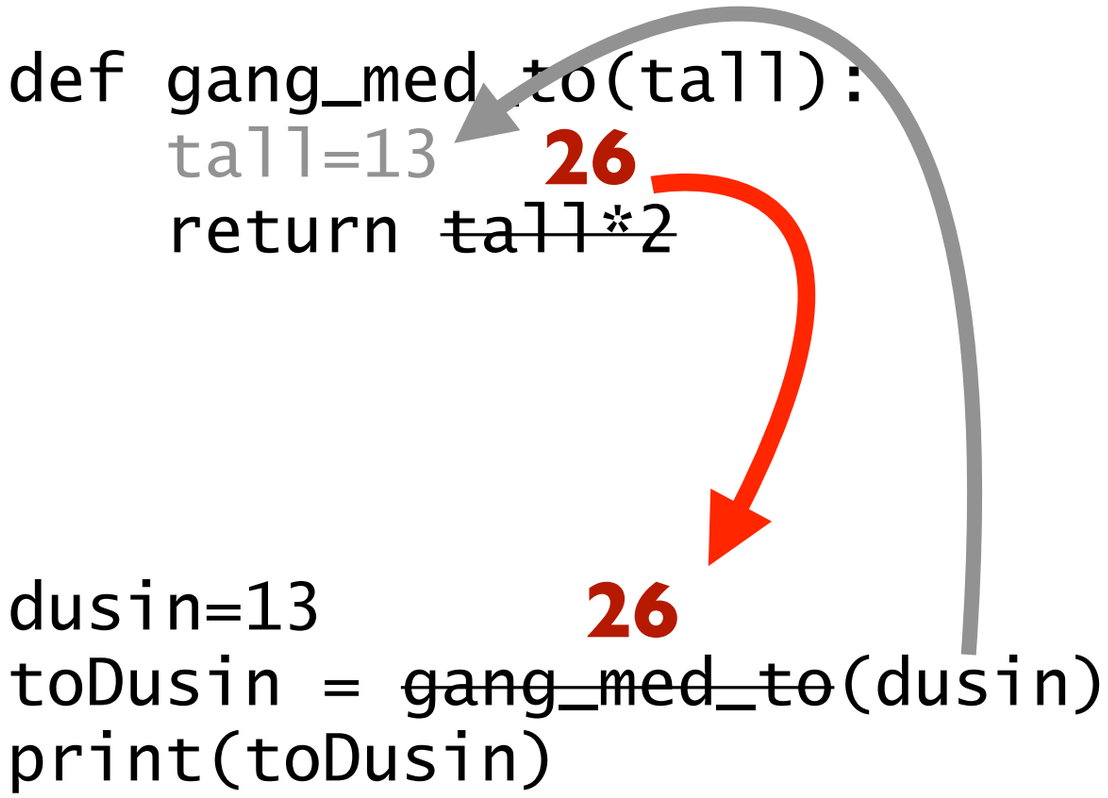
Funksjonskallet *evaluerer* til *returverdien*

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

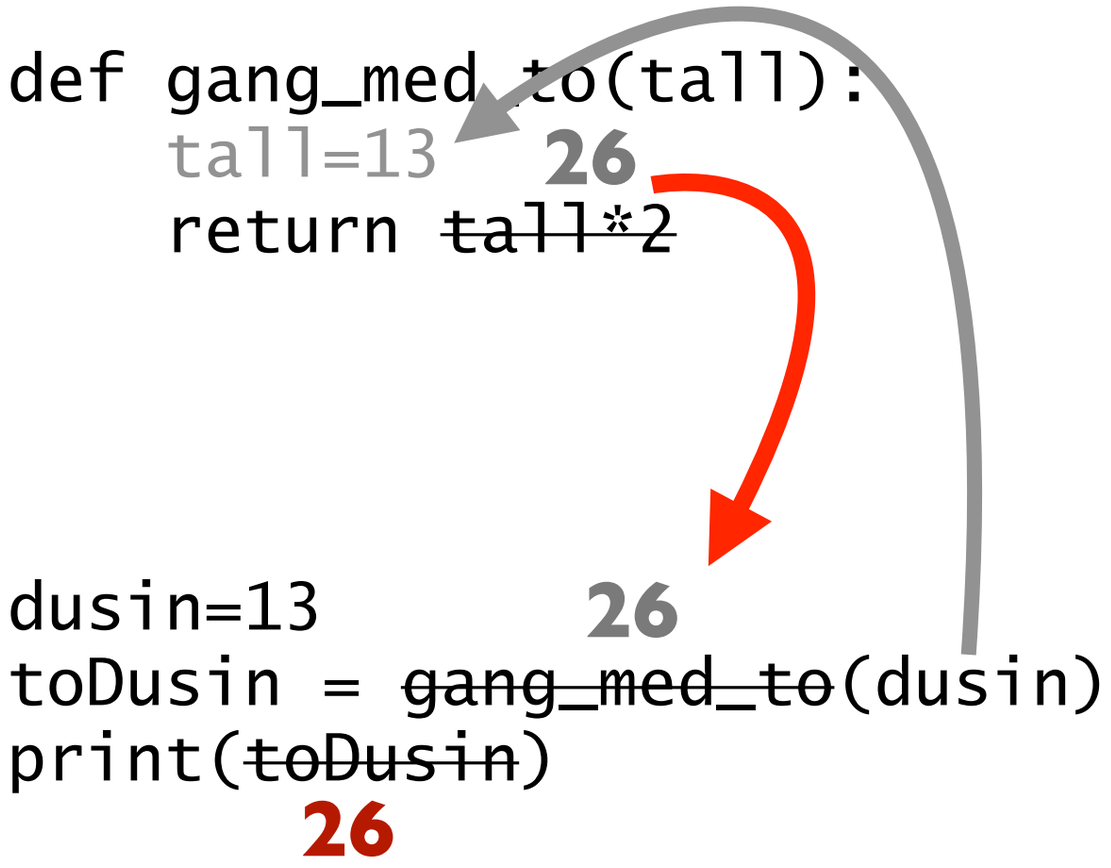
dusin=13
toDusin = gang_med_to(dusin)
print(toDusin)
```

26

Funksjonskallet *evaluerer* til *returverdien*



Funksjonskallet *evaluerer* til *returverdien*



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 synonymer 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 deklarert 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 deklarert 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..
```

Oppsummering

- Å hente data fra filer er gøy!
  - Kan jobbe med store og reelle data, uten tasting
- Formålet til en funksjon er typisk å beregne en bestemt returverdi basert på argumentene den mottar
  - Dermed inneholder funksjoner oftest ikke print og input
- Når alt dere har lært frem til nå kobles sammen, kan det brukes til å utrette mye