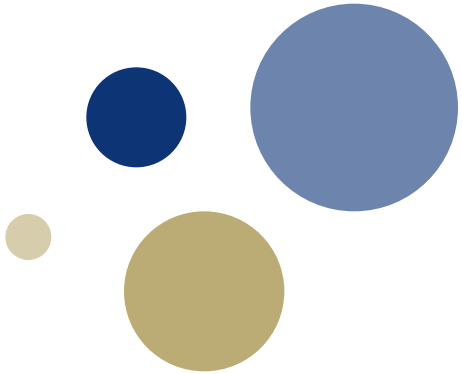
NTNU

Kunnskap for en bedre verden



Python: Filer og unntak

Gaddis: Kapittel 6

TDT4110 IT Grunnkurs

Professor Guttorm Sindre

Læringsmål og pensum

- Mål
  - Lære hva filer er
  - Lære hva unntaksbehandling er
  - Kunne bruke inn- og utoperasjoner i Python
  - Kunne programmere lesing og skriving til fil
  - Kunne bruke unntak (exceptions)
- Pensum
  - Starting out with Python, ”Files and Exceptions”

Inn- og utoperasjoner

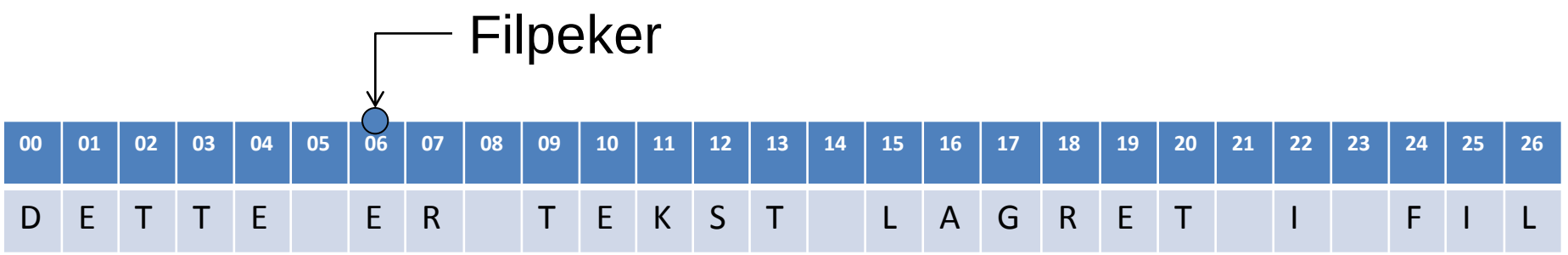
- HVORFOR trenger vi dette?
  - Hittil
    - lest inn data fra tastatur og skrevet ut til skjerm, input(), print()
      - ok for små eksempler, ubrukelig for store datamengder
    - hatt data i enkeltvariable, lister etc.
      - Ok for prosessering underveis i programutførelsen
      - Men data i primærminne glemmes når programmet avsluttes, maskinen slås av, eller lignende
- Typisk situasjon i den virkelige verden
  - Store datamengder må kunne lagres permanent
  - Data kommer fra nettet, sensorer, etc,
    - Skal ikke tastes inn manuelt men prosesseres helautomatisk
- Trenger da input/output (I/O) med filer
  - dere har brukt filer allerede (.py, .pdf, …)

Hva er ei fil i Python?

- Representert som en verdi av typen *file*
  - Denne verdien / variabelen er ikke fila, bare en referanse
- Hvorfor bruke referanser til filene?
  - Ei fil kan inneholde mye data, > minnekapasitet
    - Referansen krever mye mindre plass
      - ~ adressa til en bygning tar mye mindre plass enn selve bygningen
  - Via referansen kan vi
    - Navigere gjennom ei fil
    - Finne data vi søker etter
    - Putte deler av fila (eller hele hvis plass) inn i variable

Hva er ei fil i Python? (2)

- I ei fil lagres data etter hverandre (sekvensielt)
  - Kan sammenlignes med en tekststreng
    - Men tegnene ligger i sekundærminne (disk, minnepinne, …)
    - Fila er muterbar (kan endre innhold)
- Filpeker
  - holder orden på hvor langt man har kommet i fila.
  - kan flyttes ved kommandoer.



Filoperasjoner i Python

- Tre hovedsteg:
  1. Fila åpnes
     - Etablerer en referanse til fila
     - Filreferansen blir lagret i en variabel
     - Alle operasjoner på fila må gå via denne variabelen
  2. Verdier leses fra og skrives til fila
     - Lesing: Data lagret i fil leses inn og lagres i variable
     - Skriving: Data lagret i variable skrives til fila
  3. Fila lukkes
     - Etter at fila er lukket, kan man ikke lese eller skrive til fila
     - Poeng (bl.a.):
       - Frigjøre fila så andre kan bruke den
       - Sørge for at filbruken avsluttes på en trygg måte

Filhåndtering i Python

Filkommandoer	Forklaring
f = open(‘filnavn’)	Åpner ei fil, returnerer filreferanse
f = open(‘filnavn’,’tilgang’)	Åpner ei fil, med spesifisert tilgang. F.eks. ‘w’ åper ei fil for skriving (se neste side)
f.read()	Returnerer hele innholdet av fila
f.read(n)	Returnerer n karakterer av innholdet
f.readline()	Returnerer neste linje (før \n)
f.readlines()	Returnerer hele innholdet av fila som ei liste
f.write(s)	Skriver ***strengen*** s til fil
f.writelines(liste)	Skriver innholdet av liste av ***strenger*** til fil
f.seek(offset,fra_hvor)	Forflytter filpekeren (index) i fila
f.tell()	Returnerer posisjon til filpekeren i fila
f.close()	Lukker fila

f representerer variabelen som tar vare på filpekeren

Åpning av filer

- For å bruke ei fil må den først åpnes ved **open**:

  variabel = open(’filnavn’ , ’tilgangstype’)

- Forklaring:
  - **variabel:** Får en referanse som peker til fila med angitt filnavn
  - **filnavn**: Angir et stinavn og filnavn til fila som skal åpnes
    - Hvis fila er på samme katalog som programmet trengs ingen sti
  - **tilgangstype**: kode for typen filoverføring som skal gjøres
- Eks:

```
f = open(’datafil.txt’,’r’)                    # Åpner fil for lesing
f = open(’datafil.txt’,’w’)                    # Åpner fil for skriving
f = open(r’C:\Users\GS\textfiles\fil2.txt’,’a’)        # med sti,… legge til
# r  først inni parentesen for å unngå at \ tolkes som spesielle tegn
```

Tilgangstyper for open

- Vi har følgende tilgangstyper for `open`:

Type	Åpner for…	Tidl. innh.	Hvis fila ikke fins?
`'r'`	Lesing.	Bevares	FEIL
`'w'`	Skriving	Fjernes	Oppretter ny fil
`'a'`	Skriving (bakerst)	Bevares	Oppretter ny fil
`'r+'`	Lesing og skriving	Bevares	FEIL
`'w+'`	Lesing og skriving	Fjernes	Oppretter ny fil
`'a+'`	Lesing og skriving (bakerst)	Bevares	Oppretter ny fil

Lukking av filer

- Hvis flere programmer endrer på ei fil samtidig…
  - Kan lett bli tull
  - Operativsystemet vil derfor nekte dette
- Etter at programmet er ferdig med å bruke fila:
  - Lukk fila for å si ifra at den kan brukes av andre
  - I noen tilfeller vil skrevne data heller ikke havne på fila hvis man glemmer å lukke den til slutt
- `filvariabel.close()` `# Lukker fila`

Skrive til fil

- For å skrive data til fil i Python brukes følgende:

```
f.write(s)          # Skriver strengen s til fil med referanse f
f.writelines(liste) # Skriver en liste av strenger til fila
```

- Vi ser på et program som lar brukeren
  - Angi et filnavn (og åpne fila for skriving)
  - Gi inn tekst fra tastatur linje for linje (tom linje for å slutte)
  - …og tekstene lagres på fila
- To varianter: med write() filtest1, med writelines() filtest2

filtest1.py

filtest2.py

Viktig å merke seg!

- write(), writelines() avslutter ikke default med linjeskift
  - I motsetning til print()
  - For å skrive linjeskift i fila må vi eksplisitt legge til **'\n'**
- Kun tekststrenger kan skrives til filer
  - Andre typer data (f.eks. tall) må konverteres til strenger
  - Kan bruke **str(variabel)**
- Tilsvarende ved lesing av filer:
  - Må eksplisitt strippe '\n' fra data lest fra fil hvis vi ikke vil ha den
  - Hvis vi skal få tall, må vi konvertere fra strenger til tall
    - F.eks int( ) , float( )

Lese strenger fra fil

- For å lese strenger fra fil, benyttes:

```
streng = filvariabel.read()          # returnerer hele innholdet
                    #eller n tegn hvis parameter angis …read(n)
streng = filvariabel.readline()      # returnerer ei linje
streng = filvariabel.readlines()     # returnerer ei liste av linjene
```

- **read()** og **readlines()** bør unngås for store filer.
- **readline()** og **readlines()** krever fil delt med linjeskift (\n).
  - readline() : bruke while-løkke for å sjekke om fila er slutt
  - readlines() : lese hele i en setning, uten løkke
- Vi ser på hvordan de virker i interaktiv modus…

Å bruke Pythons for-løkke til å lese linjer

- Python tillater å skrive ei for-løkke som leser linjer fra fil og slutter ved enden av fila:
  - Format:
```
for line in file_object:
    kode..
```
  - Løkka går igjennom (itererer) fila linje for linje
- Eksempel:
  - Vi har noen filer med datoer og valutakurser
  - Ønsker å lese disse inn i liste, gjøre om kursene til tall
  - Gjøre en enkel analyse av kursutviklingen (min, max, snitt)

Kode: valutaanalyse.py

Oppgave: Lese og skrive fil

- ”Joe” vil analysere kursen på EUR vs AUD
  - Har ingen fil for dette
  - Men har filer for begge valutaene vs. NOK
- Lag et program som
  - Leser inn i lister kursene for EUR og AUD
    - Hint: bruk les_inn_kursutvikling() fra eksemplet
    - Filene heter EUR_NOK.txt og AUD_NOK.txt
  - Lager ei fil EUR_AUD.txt i tilsvarende format

Kode: eur-vs-aud1.py

Løsning: eur-vs-aud2.py

Unntak (“Exceptions”)

Kapittel 6.4

Exception / Unntak

- Under kjøring av program kan det oppstå feil
  - Programmet stopper, feilmelding i rødt
  - F.eks. divisjon på 0, for stor indeks til liste eller streng, forsøk på å addere, bruke int() på en streng med annet enn tall, …
- Unntak kan brukes generelt i Python, ikke bare for filer
  - Men brukes særlig hyppig for filer, pga mange mulige feil:
    - Forsøk på å åpne fil som ikke fins
    - Fila er under bruk av andre, kan ikke åpnes
    - Lagringsmediet er korrupt, kan ikke leses
    - Data på fil har annet format enn forventet
    - …
- try/exception kan unngå at programmet stopper opp
  - I stedet håndtere feilen på en mer elegant måte.

Exception: try – except uttrykk

- "Usikker" kode skrives inne i et **try:** uttrykk
  - Hvis alt går som det skal, er det koden inni try som kjøres
- Alternativ kode ved eventuelle feil: **except ExceptionName:**

```
try:                          # En feil i try-blokka trigger except
    uttrykk
    uttrykk
    …
except ExceptionName:         # Hopper hit hvis feil i try
    uttrykk                   # (såframt type unntak stemmer)
    uttrykk
    …
```

- Minner om et if-else-uttrykk

Enkelt eksempel UTEN filer

- int() vil feile for annen input enn tall. To løsninger:
  - Bruke if-setninger, være sikker på at vi alltid har tall til int()
  - Bruke unntak, la det feile men sørge for at det skjer elegant

```
print('Alle data skal gis som heltall.')
alder = int(input('Oppgi alder: '))
vekt = int(input('Oppgi høyde i cm: '))
h = int(input('Høyde i cm: '))
epler = int(input('Hvor mange epler har du? '))
```

```
print('Alle data skal gis som heltall.')
alder = input('Oppgi alder: ')
if alder.isdigit():
        alder = int(alder)
else:
        print('Feil format på inndata')
vekt = input('Oppgi høyde i cm: ')
if vekt.isdigit():
        vekt = int(vekt)
else:
        print('Feil format på inndata')
# Kode ikke komplett, forts. med h og epler
```

```
print('Alle data skal gis som heltall.')
try:
        alder = int(input('Oppgi alder: '))
        vekt = int(input('Oppgi høyde i cm: '))
        h = int(input('Høyde i cm: '))
        epler = int(input('Hvor mange epler har du? '))
except ValueError:
        print('Feil format på inndata')
```

Exception – ExceptionName

- Ulike typer Exceptions har ulike navn, f.eks.
  - ValueError: feil data for funksjon, f.eks. **int('ost')**
  - TypeError feil type data for operasjon, f.eks **2+'ost'**
  - ZeroDivisionError: prøver å dividere med 0
  - IndexError: feil indeks, f.eks a=[8,3,2], … **x=a[4]**
  - OSError Feil ved bruk av OS-funksjoner, f.eks filer
    - IOError er f.o.m. Python v 3.3 bare en alias for denne
    - Flere undertyper: FileNotFoundError, FileExistsError, …
  - Exception: Alle mulige feil (generell)
- Vi tuller litt med fila AUD.txt og ser hvordan dette påvirker programmet vårt

Kode: eur-vs-aud2.py, eur-vs-aud3.py

Exception – vis feilmelding

- Det er mulig å vise feilmeldingen som Python gir:

```
try:
    uttrykk…
except  Exception as variabel:
    …
    print(variabel)
```

- Uttrykket as variabel
  - lagrer feilmeldingen i variabelen

eur-vs-aud4.py

Exception – else og finally

Et try – except uttrykk kan også bestå av else og finally:

else blir utført hvis ingen exceptions ble trigget.

finally blir utført til slutt, uansett om exceptions ble trigget eller ikke

```
try:
    uttrykk...
except ExceptionName:
    uttrykk...
else:
    uttrykk...
finally:
    uttrykk...
```

eur-vs-aud5.py

Filhåndteringsprosess

- Tre hovedsteg:
  1. Åpne fil med en gitt aksess
  2. Lese fra /skrive til fila, evt. forflytte filpeker
  3. Lukke fila
- Vi kan jobbe med flere filer samtidig:
  - Filvariabelen med referanse til fila bestemmer hvilken fil vi jobber med.
  - Kan f.eks. la bruker skrive inn filnavn fra tastaturet
    - Velge mellom alternative filer for lesing
    - Eller alternative filer å skrive til

Exceptions / Unntak

- Exception: feil som skjer når et program kjører
  - Som regel fører det fører det til at programmet stopper (kræsjer, feilmelding)
- Exception handling: Håndtere ”exceptions” ved å kjøre alternativ kode heller enn å kræsje
- Benytter:

```
try:                    # Prøv om koden lar seg kjøre
except                  # Fanger opp hvis koden i try feiler
except Exception as variable:  # fanger feilmelding
else:                   # Kjøres hvis det ikke blir exception
finally:                # Kjøres uansett til slutt
```