

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?

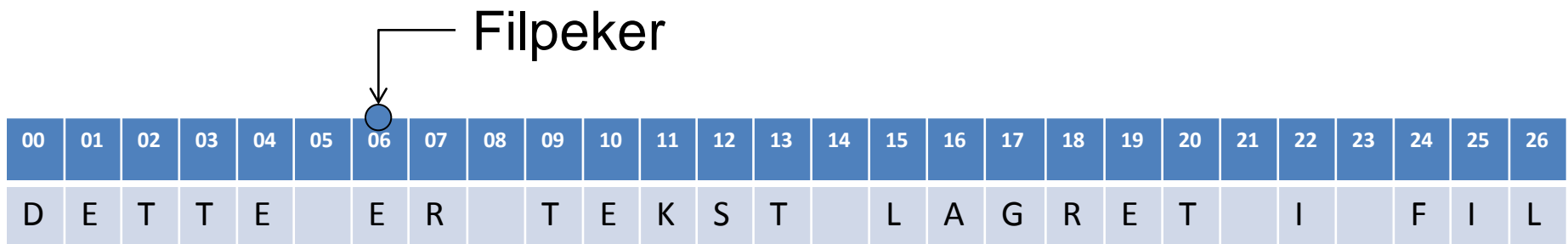


- Representert som en verdi av typen *file*
 - Denne verdien / variabelen er ikke fila, bare en referanse
- Hvorfor bruke referanser til filene?
 - Ei fil kan inneholde mye data, > minnekapasitet
 - Referansen krever mye mindre plass
 - ~ adressa til en bygning tar mye mindre plass enn selve bygningen
 - 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
 - Skrivning: 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
<code>f = open('filnavn')</code>	Åpner ei fil, returnerer filreferanse
<code>f = open('filnavn', 'tilgang')</code>	Åpner ei fil, med spesifisert tilgang. F.eks. 'w' åpner ei fil for skriving (se neste side)
<code>f.read()</code>	Returnerer hele innholdet av fila
<code>f.read(n)</code>	Returnerer n karakterer av innholdet
<code>f.readline()</code>	Returnerer neste linje (før \n)
<code>f.readlines()</code>	Returnerer hele innholdet av fila som ei liste
<code>f.write(s)</code>	Skriver strengen s til fil
<code>f.writelines(liste)</code>	Skriver innholdet av liste av strenger til fil
<code>f.seek(offset, fra_hvor)</code>	Forflytter filpekeren (index) i fila
<code>f.tell()</code>	Returnerer posisjon til filpekeren i fila
<code>f.close()</code>	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 Python's 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



LETTERE:

Start med fila `prog.txt` og koden `ps_lett_V0.py`.

- (a) Lag funksjonen `les_fil()` for å lese fra fila og inn i ei liste som returneres fra funksjonen
- (b) Lag funksjonen `skriv_fil()` som skriver ei liste til fil

MIDDELS:

Start med fila `prog2.txt` og koden `ps_mid_V0.py`

- (a) Lag funksjonen `les_fil()`
- (b) Lag funksjonen `skriv_fil()`

VANSKELIG:

Se på [gamle_valutakurser.txt](#) som er hentet fra Norges Bank sine websider

Les denne fila inn i ei 2D-liste hvor tekst lagres som strenger, heltall som int og desimaltall som float.

NB: Merk at første linje av fila har andre slags data enn resten, og at det står 'ND' (No Data) en del steder hvor det normalt ville ha stått tall. Merk også at fila har komma i desimaltall (norsk standard) mens Python trenger punktum for float



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, feil indeks til liste, int() på en streng med annet enn tall, ...
- Unntak kan brukes generelt i Python, ikke bare for filer
 - Men særlig vanlig for filer, pga mange mulige feil:
 - Forsøk på å åpne fil som ikke fins
 - Fila er i bruk av andre, kan ikke åpnes
 - Lagringsmediet er korrupt, kan ikke leses
 - Data på fil har annet format enn forventet
 - ...
- **try/except** kan unngå at programmet stopper opp
 - 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
    ...
```

- Hvis navn på unntaket droppes, tar den alle unntak
- Minner om et if-else-uttrykk (hvis ok så ... ellers ...)

Enkelt eksempel UTEN filer

- `int()` vil feile for annen input enn tall. To løsninger:
 - Bruke `if-setninger`, sikre at vi alltid har tall til `int()`
 - Bruke `unntak`, la `int()` feile men takle feilen

```
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)
 - Dette er også hva man får hvis man bare skriver `except`:

Hva bør try-blokka dekke?



- Koden som kan gå galt
- PLUSS: kode som det ikke gir mening å kjøre hvis forutgående kode gikk galt
- Vi ser videre på middels-oppgaven fra i sted
 - Sette try-blokk bare i funksjonen `les_fil()` ?
 - Ikke så god ide, for hvis det feiler...
 - ...blir variabelen `tabell` i `main()` uten innhold
 - Virker derfor lurene å sette try-blokka i `main()`

KODE: `ps_mid_v2.py`
`ps_mid_v3.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

ps_mid_v4.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...
```

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