NTNU

Kunnskap for en bedre verden

Kunnskap for en bedre verden

TDT4110 Informasjonsteknologi grunnkurs:
Python: Repetisjon
tekstfiler
rekursjon

Terje Rydland - IDI/NTNU

Filbehandling
Tekstfiler

Prosessen for filoperasjoner i Python

- Filoperasjoner i Python gjøres i tre steg (uavhengig av om det er tekstfiler eller binærfiler):
  1. Fila åpnes: Etablerer en link mellom filvariabelen og informasjonen lagret på disken.
     - Filreferansen som peker til den fysiske fila på disk blir lagret i en variabel.
     - Alle operasjoner mot fila må bruke denne variabelen som filreferanse.
  2. Verdier leses fra og skrives til fila ved hjelp av filreferansen:
     - Lesing: Data lagret i fil leses inn og lagres i variabler.
     - Skriving: Data lagret i variabler skrives som data lagret i en fil.
  3. Fila stenges.
     - Etter at fila er stengt, kan man ikke lese eller skrive til fila.

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' åpner 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()	Stenger fila

f representerer variabelen som tar vare på filpekeren

Søk igjennom stor fil

- Skriv funksjonen *find_rate* som har tre input-parametere, *filname*, *check_rate*, og *acc*.
- Funksjonen skal søke igjennom en fil som inneholder vekslingsrate mellom USD og NOK sammen med dato, og skrive ut til skjerm alle datoer der vekslingsraten i fila matcher med *check_rate*. Fila (USD_NOK.txt) har følgende format:

  26.11.2013 6.1036

  25.11.2013 6.1236

  ...

- Parameteren *acc* brukes til å angi hvor mange siffer det avrundes til under sammenligning av vekslingsrater.

Hver linje ser altså slik ut:

'26.11.2013 6.1036\n'
'25.11.2013 6.1236\n'
etc...

kode: find_rate.py

Exception...

- Utvid funksjonen *find_rate* slik funksjonen tåler kløne brukere:
  - Hvis man kaller *find_rate* med et filnavn til ei fil som ikke finnes eller som ikke kan leses, så skal følgende skrives ut til skjermen: ”Feil: Finner ikke fila eller har ikke lesetilgang!”
  - Hvis man oppgir feil *check_rate* eller *acc*, f.eks. tekststrenger i stedet for tall, så skal følgende skrives ut til skjermen: ”Feil: Det er noe feil med argumentene!”)

kode: find_rate_try.py

Rekursjon

NTNU
Det skapende universitet

Rekursjon

- Rekursiv funksjon kalles det når en funksjon kaller seg selv.
- Rekursive funksjoner brukes ofte i algoritmer for å lage elegante løsninger på problemer.
- For at en rekursiv funksjon ikke skal fortsette i det uendelige, er det viktig at argumentet til funksjonen endres slik at den en gang får en stopp verdi.

rekursjon.py - /Users/terjery/Library/Mobile Documents/com~apple~CloudDo...

```
def count_down(antall):
    if (antall>0):
        print("Teller ned:",antall)
        count_down(antall - 1)


count_down(5)
```

Ln: 1 Col: 0

rekursjon.py

NB

- Noe som kan løses rekursivt kan også **alltid** løses iterativt

rekusjon.py - /Users/terjery/Library/Mobile Documents/com~apple~CloudDocs/ITGK...

```
def count_down(antall):
    if antall>0:
        print('Teller ned:',antall)
        count_down(antall-1)

count_down(5)
```

Ln: 7 Col: 0

tell_ned.py - /Users/terjery/Library/Mobile Documents/com~apple~CloudD...

```
def count_down(antall):
    for i in range(antall,0,-1):
        print(i)

count_down(5)
```

Ln: 6 Col: 0

Rekursjon til å kalkulere fakultet

- Fakultet til et tall beregnes på følgende måte:
  - Hvis n=0 så er n!=1
  - Hvis n>0 så er n!=1 * 2 * 3 * … * n
- For å lage en algoritme for fakultet må:
  - Først finne delen som ikke er rekursiv (som ikke skal kalles på nytt):
    - Hvis n = 0 så er n!=1 (denne er ikke avhengig av tidligere ledd)
  - Resten er den rekursive delen som stegvis kan beskrives:
    - Hvis n>0 så er fakultet(n) = n x fakultet(n-1)
  - Dette er alt vi trenger for å lage en rekursiv funksjon for å beregne fakultet

fakultet.py

fakultet.py

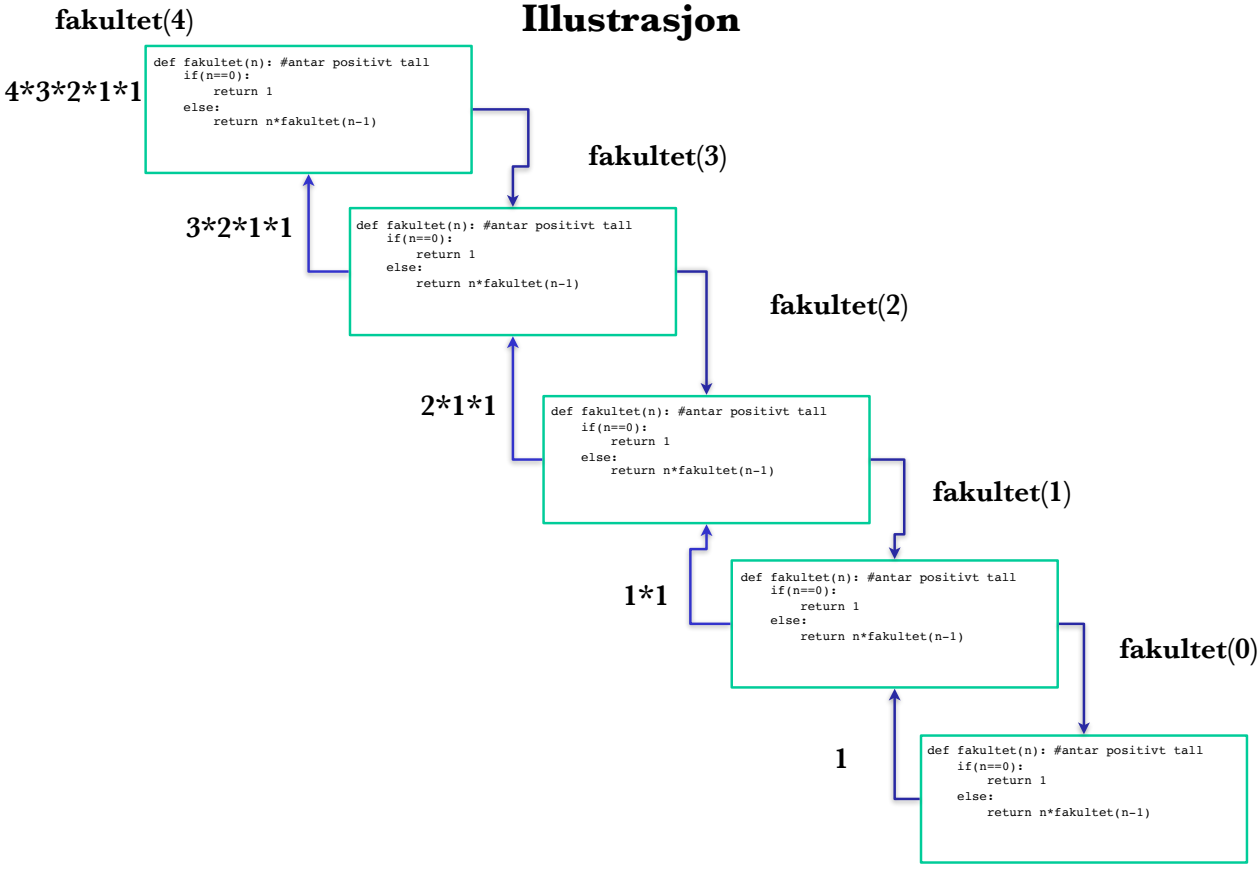
fakultet.py - /Users/terjery/Library/Mobile Documents/com~apple~C...

```
def fakultet(n):
    if(n==0):
        return 1
    else:
        return n * fakultet(n-1)

print(fakultet(4))
```

Ln: 1 Col: 0

Illustrasjon



fakultet.py

Rekursivt

fakultet.py - /Users/terjery/Library/Mobile Documents/com~apple~C...

```
def fakultet(n):
    if(n==0):
        return 1
    else:
        return n * fakultet(n-1)

print(fakultet(4))
```

Ln: 1 Col: 0

Iterativt

fakultet.py - /Users/terjery/Library/Mobile Documents/com~apple~CloudDo...

```
def fakultet(n):
    prod = 1
    for i in range(1,n+1):
        prod *=i
    return prod

print(fakultet(4))
```

Ln: 8 Col: 0

Binærsøkealgoritmen

- Binærsøkealgoritmen fungere på samme måte som vi kan spille gjettespillet fra uke 39.
  - Gjetter på den midterste verdien, og fjerner dermed halvparten av mulighetene for hver gjetting.

gjettespill.py

Illustrasjon av binærsøk

Finnes 17 i listen?

indeks	0	1	2	3	4	5	6	7	8	9
verdi	1	2	3	9	11	13	17	25	57	90

Illustrasjon av binærsøk

Finnes 17 i listen?

indeks	0	1	2	3	4	5	6	7	8	9
verdi	1	2	3	9	11	13	17	25	57	90

Sjekk det midterste elementet. **(0 + 9)//2** **->** **4, altså a(4)**

Illustrasjon av binærsøk

Finnes 17 i listen?

indeks	0	1	2	3	4	5	6	7	8	9
verdi						13	17	25	57	90

Sjekk det midterste elementet. (0 + 9)//2 -> 4, altså a(4)

17 > a(4)

Sjekk da det midterste elementet mellom a(5) og a(9) (5 + 9)//2 -> 7

15

Illustrasjon av binærsøk

Finnes 17 i listen?

indeks	0	1	2	3	4	5	6	7	8	9
verdi						13	17	25		

Sjekk det midterste elementet. (0 + 9)//2 -> 4, altså a(4)

17 > a(4)

Sjekk da det midterste elementet mellom a(5) og a(9) (5 + 9)//2 -> 7

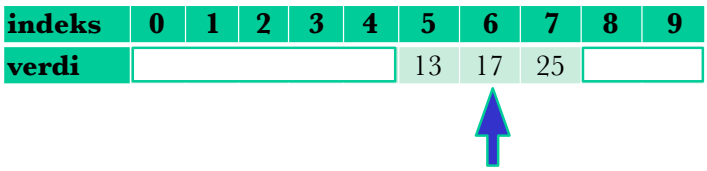
17 < a(7)

Sjekk da det midterste elementet mellom a(5) og a(7) (5 + 7)//2 -> 6

NTNU

Illustrasjon av binærsøk

Finnes 17 i listen?



Sjekk det midterste elementet. (0 + 9)//2 -> 4, altså a(4)

17 > a(4)

Sjekk da det midterste elementet mellom a(5) og a(9) (5 + 9)//2 -> 7

17 < a(7)

Sjekk da det midterste elementet mellom a(5) og a(7) (5 + 7)//2 -> 6

17 == a(6)

17 finnes i listen

Rekursiv algoritme for binærsøk

```
bin_search(liste, verdi, min, max)
  hvis max < min Returner ikke funnet verdi
  hvis ikke
    midtpunkt = midtpunkt(min,max)
    hvis verdi < liste[midtpunkt]
       returner bin_search(liste,verdi,min,midtpunkt-1)
    hvis verdi > liste[midtpunkt]
       returner bin_search(liste,verdi,midtpunkt+1,max)
    hvis ikke
       returner midtpunkt
```

binary_search.py

binary_search.py

binary_search.py - /Users/terjery/Library/Mobile Documents/com~apple~CloudDocs/ITGK 2016/Python/Keynote/Uke 45/Python/binary_search.py (3.5.1)

```
# Name: binary_search
# Input: a list and a search value
# Ouput: True if searchValue is found, else False
# Description: Search through a sorted list using the binary search algorithm

def binary_search(liste, verdi, imin, imax):
    if(imax < imin):
        return False
    else:
        imid = (imin+imax)//2 # Midtpunktet
        if (verdi<liste[imid]):
            return binary_search(liste,verdi,imin,imid-1)
        elif (verdi>liste[imid]):
            return binary_search(liste,verdi,imid+1,imax)
        else:
            return imid

A = [1,2,3,9,11,13,17,25,57,90]
result = binary_search(A,57,0,len(A)-1)
print(result)
```

Ln: 1 Col: 0

Karakteristikker av binærsøk

Karakteristikker av binærsøk

- Hva er best-case?
  - Treffer på midten første gang
  - Tidsbruk: $\Omega(1)$

18

Karakteristikker av binærsøk

- Hva er best-case?
  - Treffer på midten første gang
  - Tidsbruk: $\Omega(1)$
- Hva er worst-case?
  - Halverer lista helt til det er ett element igjen
  - Tidsbruk: $O(\log n)$

NTNU

Karakteristikker av binærsøk

- Hva er best-case?
  - Treffer på midten første gang
  - Tidsbruk: $\Omega(1)$
- Hva er worst-case?
  - Halverer lista helt til det er ett element igjen
  - Tidsbruk: $O(\log n)$
- Kan sekvensielt søk være raskere enn binærsøk?
  - Ja hvis sekvensielt søk treffer tidlig i lista

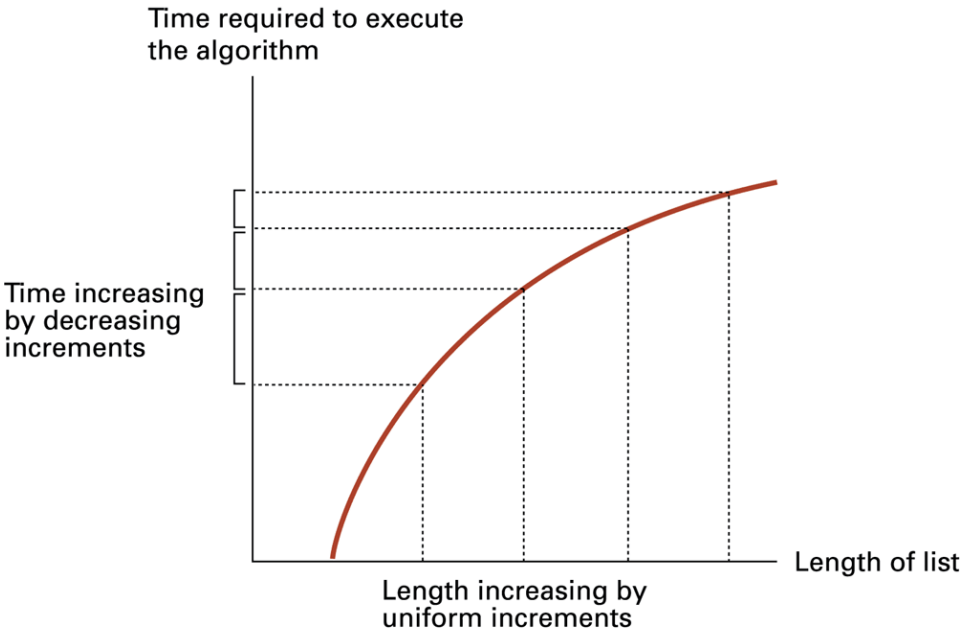
18

Karakteristikker av binærsøk

- Hva er best-case?
  - Treffer på midten første gang
  - Tidsbruk: $\Omega(1)$
- Hva er worst-case?
  - Halverer lista helt til det er ett element igjen
  - Tidsbruk: $O(\log n)$
- Kan sekvensielt søk være raskere enn binærsøk?
  - Ja hvis sekvensielt søk treffer tidlig i lista
- Begrensninger på binærsøk:
  - Lista må være sortert (pre-prosessert)

NTNU

Graf over worst-case analyse av binært søk



GCD - Greatest Common Divisor

- Euclids algoritme bruker
  - divisjon og observasjonen at GCD mellom 2 tall er også en faktor i forskjellen mellom tallene
  - gcd(48,18)
    - divider 48 med 18. Dette gir 2 med en rest på 12
    - divider 18 med 12. Dette gir 1 med rest 6.
    - divider 12 med 6. Dette gir en rest på 0, hvilket betyr at 6 er gcd.
- Euclids algoritme for å finne største fellesnevner for 2 tall skrevet litt mer formelt:
  - gcd(a,0) = a
  - gcd(a,b) = gcd(b, a mod b)

Som Python-program

```
# gcd funksjonen returnerer den største fellesnevneren til to tall

def gcd(x, y):
    if x % y == 0:
        return y
    else:
        return gcd(y, x % y)
```

Rekursivt

gcd.py - /Users/terjery/Library/Mobile Documents/com~apple~CloudDocs/ITGK 2016/Python/1Forelesni...

```
def gcd(x, y):
    if x > y:
        storst = x
        minst = y
    else:
        storst = y
        minst = x
    teller = minst
    gcd = 1
    while True:
        if storst % teller == 0 and minst % teller == 0:
            break
        else:
            teller -=1
    return teller
```

Ln: 27 Col: 0

Iterativt

Eks - tallene 18 og 48

- gcd(48,18)
  - return gcd(18,12)
    - return gcd(12, 6)
  - 6

```
# gcd funksjonen returnerer den største fellesnevneren til to tall

def gcd(x, y):
    if x % y == 0:
        return y
    else:
        return gcd(y, x % y)
```

```
def gcd(x,y)        x = 48, y = 18
    if x % y == 0:
        return y
    else:
        return gcd(y, x % y)
```

```
def gcd(x,y)        x = 18, y = 12
    if x % y == 0:
        return y
    else:
        return gcd(y, x % y)
```

```
def gcd(x,y)        x = 12, y = 6
    if x % y == 0:
        return y
    else:
        return gcd(y, x % y)
```

Eks - tallene 18 og 48

- gcd(48,18)
  - return gcd(18,12)
    - return gcd(12, 6)
  - 6

```
# gcd funksjonen returnerer den største fellesnevneren til to tall

def gcd(x, y):
    if x % y == 0:
        return y
    else:
        return gcd(y, x % y)
```

```
def gcd(x,y)      x = 48, y = 18
    if x % y == 0:
        return y
    else:
        return gcd(y, x % y)
```

48 % 18 != 0, så gcd blir kalt med verdiene 18 og 12

```
def gcd(x,y)                x = 18, y = 12
    if x % y == 0:
        return y
    else:
        return gcd(y, x % y)
```

Som så returneres til kallende funksjon

18 % 12 != 0, så gcd blir kalt med verdiene 12 og 6

```
def gcd(x,y)                x = 12, y = 6
    if x % y == 0:
        return y
    else:
        return gcd(y, x % y)
```

12 % 6 == 0, så verdien 6 blir returnert