NTNU
Kunnskap for en bedre verden

Institutt for datateknikk og informasjonsvitenskap

Løsningsforslag til TDT4110 IT–grunnkurs, med Python

Eksamensdato: **2016-12-06**

Hjelpemiddelkode/Tillatte hjelpemidler: **D – Bare bestemt, enkel kalkulator**

Innhold:

- Oppgave 1: Flervalgsoppgave (25%)
- Oppgave 2: Kodeforståelse (20%)
- Oppgave 3: Programmering Valg (25%)
- Oppgave 4: Programmering Penger (30%)

Oppgave 1: Flervalgsoppgave (25%)

1. c
2. c
3. d
4. c
5. c
6. b
7. a
8. d
9. a
10. b
11. c
12. c
13. c
14. b
15. a
16. a
17. c
18. b
19. d
20. c
21. d
22. b
23. b
24. d
25. a

Oppgave 2: Kodeforståelse (20%)

Oppgave 2a (5%)

Følgende linje blir printet ut:

```
[[1, 0, 1], [0, 1, 0], [1, 0, 1]]
```

`flopp` tar inn en todimensjonal liste (matrise) og gjør om alle enere til 0, og alle andre verdier til 1.

Oppgave 2b (5%)

Følgende linje blir printet ut:

```
30th Nov 1337
```

`compute` tar inn tre parametere som inneholder dag, måned og år, og returnerer en korrekt formatert engelsk dato-tekst tilbake.

Oppgave 2c (5%)

Følgende linje blir printet ut:

```
3 b
```

`fr` tar inn en tekst og finner ut hvilken bokstav det er flest av i teksten, og returnerer to verdier: antall forekomster av bokstaven samt bokstaven selv.

Oppgave 2d (5%)

Følgende linje blir printet ut:

```
6
```

`f` regner ut den rekursive tverrsummen tallet som den får inn som parameter, dvs. fortsetter å ta tverrsum helt til vi får et ensifret tall. (Tverrsummen av 32145 blir 15, tverrsummen av 15 blir deretter 6)

Oppgave 3: Programmering Valg (25%)

Antar i denne oppgaven at vi har antall distrikt samt lista med partinavn tilgjengelig som globale variable. Om man i stedet tar det inn som parameter eller definerer disse verdiene lokalt i funksjoner er det også ok.

```
DISTRICTS = 92
parties = ['Tea Party', 'Coffee Party', 'Milk Party',
           'House Party', 'Beach Party']
```

Oppgave 3a: Initialisering (5%)

Løsning med "list comprehension":

```
def initElection(parties):
    return [ [ 0 for i in range(len(parties)) ]
                 for j in range(DISTRICTS) ]
```

Løsning med løkke som oppretter en og en rad (flere andre måter også mulig):

```
def initElection(parties):
    election = [ ]
    for i in range(DISTRICTS):
        election.append([0]*len(parties))
    return election
```

Oppgave 3b: Oppdatering (5%)

```
def updateElection(election, district, votes):
    for i in range(len(votes)):
        election[district][i] += votes[i]
    return election
```

Oppgave 3c: Presidentvalget (5%)

Lager her to hjelpefunksjoner, sumVotes() som returnerer ei enkel liste som er summen av stemmer for hvert parti for hele landet, og leadIndex() som returnerer indeksen til det høyeste av stemmetallene i ei slik enkel liste. Sistnevnte funksjon er tenkt gjenbrukt i 3d.

```
def sumVotes(election):
    total = [0] * 5
    for row in election:
        for i in range(len(row)):
            total[i] += row[i]
    return total

def leadIndex(votes):
    lead = 0
    for i in range(1, len(votes)):
        if votes[i] > votes[lead]:
            lead = i
    return lead
```

```
def printLeadP(election):
    sums = sumVotes(election)
    lead = leadIndex(sums)
    print(parties[lead], 'is leading the election with',
          sums[lead], 'votes')
```

Alternativ løsning. Denne benytter metoden index() for å finne indeksen til høyeste stemmetall, så man slipper noen løkke à la hjelpefunksjonen leadIndex() over.

```
def printLeadP(election,parties):
      summ = [0]*len(parties)
      for i in range(len(election)):
            for j in range(len(parties)):
                  summ[j] += election[i][j]
      storst = max(summ)
      indeks = summ.index(storst)
      print(parties[indeks],'is leading the election with',
            storst,'votes.')
```

Oppgave 3d: Parlamentsvalget (10%)

Lager en hjelpefunksjon sumDelegates() som returnerer ei liste med antall delegater for hvert parti, samt antall valgkretser som er "tied" og antall med "no votes". Lista d i denne hjelpefunksjonen opprettes derfor med 2 ekstra elementer (jfr. len(election[0]) **+ 2**) for å lagre antall "tied" og antall "no votes" i disse to siste elementene. Likeledes lages ei ny liste t som inneholder partinavnene pluss tekster for "tied" og "no votes", så vi kan kjøre hele utskrifta i samme løkke.

```
def sumDelegates(election):
    d = [0] * (len(election[0]) + 2)
    for row in election:
        if sum(row) == 0:           # no votes yet
            d[-1] += 1              #   count in last element
        else:
            i = leadIndex(row)      # from 3c
            if row[i] in row[i+1:]: # max also further right
                d[-2] += 1          #   count as tie
            else:                   # only one winner
                d[i] += 1           #   count for party i
    return d

def printResults(election):
    nd = sumDelegates(election)
    t = parties + ['Undecided (tied)'] + ['Undecided (no votes)']
    for i in range(len(t)):
        print((t[i]+':').ljust(30), format(nd[i], '3d'), end=' ')
        if nd[i] == 1:
            print('delegate')
        else:
            print('delegates')
```

Alternativ løsning:

```
def tekst(tall):
      if tall == 1:
            return 'delegate'
      else:
            return 'delegates'

def skrivUt(delegater,tied,noVotes,parties):
      for i in range(len(delegater)):
          print((parties[i]+':').ljust(25),
                str(delegater[i]).rjust(16),
                tekst(delegater[i]))
      print('Undecided (tied):'.ljust(25),
             str(tied).rjust(16),tekst(tied))
      print('Undecided (no votes):'.ljust(25),
             str(noVotes).rjust(16),tekst(noVotes))

def printResults(election,parties):
      delegater = [0]*len(parties)
      tied, noVotes = 0,0
      for i in range(len(election)):
            a = sorted(election[i])
            if sum(election[i]) == 0:
                  noVotes += 1
            elif a[3] == a[4]:
                  tied += 1
            else:
                  storst = max(election[i])
                  indeks = election[i].index(storst)
                  delegater[indeks] += 1
      skrivUt(delegater,tied,noVotes,parties)
```

Oppgave 4: Programmering Penger (30%)

Oppgave 4a: 2-digit (7%)

For 4a er naturlig fremgangsmåte å slå opp i dictionary D, hvis tallet fins der, kan vi returnere tilhørende tekst direkte, og ellers må vi sette sammen to tekster fra D med bindestrek mellom. Oppslag kan enten gjøres med D.get(tt) som vil returnere None hvis tt ikke fins som nøkkel i D (eller vi kan definere annen default enn None hvis vi vil), eller med D[tt] som vil feile hvis tt ikke fins som nøkkel. Mulig løsning med D.get(tt) :

```
def i2_txt(tt):
    if D.get(tt) == None:
        return D.get(tt//10 * 10) + '-' + D.get(tt % 10)
    else:
        return D.get(tt)
```

Mulig løsning med D[tt] (her må vi først teste om nøkkelen fins der, siden forsøk på å slå opp ikke-eksisterende nøkkel vil feile):

```
def i2_txt(tt):
    if tt in D.keys():
        return D[tt]
    else:
        last = tt % 10
        return D[tt - last] + '-' + D[last]
```

Som det står i oppgaveteksten, er det ikke noe krav å bruke dictionary D, så andre korrekte løsninger skal også gis full pott. Ett eksempel kan være å lage to lister i stedet:

```
def i2_txt(tt):
    a = ['zero', 'one', 'two', 'three', 'four', 'five', 'six',
         'seven', 'eight', 'nine', 'ten', 'eleven', 'twelve',
         'thirteen', 'fourteen', 'fifteen', 'sixteen',
         'seventeen', 'eighteen', 'nineteen', 'twenty']
    b = ['zero', 'ten', 'twenty', 'thirty', 'forty', 'fifty',
         'sixty', 'seventy', 'eighty', 'ninety']
    if tt <= 20:
        return a[tt]
    else:
        tens = tt // 10
        ones = tt % 10
        return b[tens] + '-' + a[ones]
```

Et annet eksempel kan være å lage bare én liste med tekster for alle tallene fra null til nittini, hvor man deretter kan få returnert riktig tekst simpelthen ved å bruke innparameteren tt som indeks til denne lista (evt. tallene 1-99 og bruke tt-1 som indeks). I kode hvor det inngår en veldig lang liste eller veldig lang if-elif-struktur, er det greit om kandidaten har vist begynnelsen og slutten, og indikert resten med … eller lignende, så lenge det klart fremgår hva som skal inn for … og dette ville ha gitt riktig kode.

Oppgave 4b: 3-digit (7%)

I denne oppgaven gjør man det lettere for seg selv ved å gjenbruke funksjonen fra 4a. Først må man finne ut om man har noen hundre, så om man skal ha et blankt tegn (mellomrom hvis man har både hundre(r) og noe mer etterpå), og til slutt kan man bruke funksjonen i2_txt() på resten.

```
def i3_txt(ttt):
    text = ''
    rest = ttt % 100
    if ttt >= 100:
        text += D[ttt//100] + ' hundred'
    if ttt >= 100 and rest > 0:
        text += ' '
    if rest > 0:
        text += i2_txt(rest)
    return text
```

Oppgave 4c: 9-digit (7%)

I denne oppgaven lønner det seg å gjenbruke funksjonen i3_txt() fra 4b. Denne kan brukes tre ganger, da vi potensielt har et tresifret tall, så ordet 'million', så et tresifret tall igjen, så ordet 'thousand', så et tresifret tall igjen, og så ingenting (eller dvs. tom streng). Ved å bruke lista L kan vi gjøre dette i ei løkke:

```
def i9_txt(num):
    text = ''
    for i in range(0, len(L), 2):
        if num >= L[i]:
            text += ' ' + i3_txt(num//L[i]) + L[i+1]
        num = num % L[i]
    return text.strip()
```

Alternativ løsning uten løkke (og uten å bruke lista L), gjør essensielt det samme. Det var ikke noe krav å bruke lista L, så løsninger à la dette bør gi like god uttelling:

```
def i9_txt(num):
    text = ''
    if num >= 1000000:
        text += i3_txt(num // 1000000) + ' million '
        num = num % 1000000
    if num >= 1000:
        text += i3_txt(num // 1000) + ' thousand '
        num = num % 1000
    if num >= 1:
        text += i3_txt(num)
    return text.strip()
```

I begge disse foreslåtte løsningene brukes strip() helt til slutt for å fjerne et eventuelt overflødig blankt tegn (som vi kan ha lagt til for tall som ikke hadde enere, slik som 9800000). Alternativt kan man ha eksplisitte if-setninger for å sjekke om en blank skal inn mellom millioner og tusener og mellom tusener og resten, a la **if ttt >= 100 and rest > 0** i 4b, men strip() fremsto her som programmeringsmessig enklere.

Oppgave 4d: Sett inn tekst (9%)

Også denne kan gjøres på flere måter, uansett vil det lønne seg å gjenbruke funksjonen i9_txt() fra 4c. Den mest fornuftige begynnelsen er å splitte lista i enkeltord og så gå i løkke og se på hvert ord om det er et tall, og da legge inn tekst for tallet.

Ett eksempel: bruke **try** med konvertering med **int()**, hvis dette går, legge til tekst. Hvis **int()** feiler, gjør vi bare **pass** fordi vi skal beholde ordet uendret[1]. I dette eksemplet jobber vi direkte i lista og får ut en full setning til slutt ved å bruke join() rundt mellomrom for returverdien.

```
def add_words(text):
    wl = text.split()
    for i in range(len(wl)):
        try:
            num = int(wl[i])
            wl[i] += ' - ' + i9_txt(num) + ' -'
        except:
            pass                # do nothing
    return ' '.join(wl)
```

Annet eksempel på løsning: bruke en test **if**... **isdigit()** for å se om det er tall. I dette eksemplet bygger vi en tekststreng underveis i løkka (alternativ til join()). Bruker strip() helt til slutt på samme måte som i 3c.

```
def add_words(text):
    new_text = ''
    wl = text.split()
    for word in wl:
        new_text += word + ' '
        if word.isdigit():
            new_text += '- ' + i9_txt(int(word)) + ' - '
    return new_text.strip()
```

Hvis man verken kom på å bruke **try** eller **isdigit()** for å sjekke om et ord egentlig var et tall, er det også helt ok å gå i en for-løkke gjennom ordet og sjekke tegn for tegn om alle er siffer, evt. som en hjelpefunksjon. Eks. (da skrive **if all_digits(word):** i koden like over i stedet for **if word.isdigit()**:):

```
def all_digits(word):
    for c in word:
        if c not in '0123456789':
            return False
    return True
```

[1] Om man ikke kom på instruksjonen **pass**, vil en hvilken som helst programsetning som ikke gjør noen skade, også kunne brukes, for eksempel x = 0. **try** uten **except** eller en helt tom **except**-del vil dessverre gi syntaksfeil, men hvis noen har gjort dette, bør det gi minimalt trekk da det likevel var nær ved å være en bra løsning av problemet og man fort ville ha sett og rettet denne syntaksfeilen hvis man jobbet i en programeditor.