Forside

Institutt for datateknologi og informatikk

Eksamensoppgave i TDT4109 - Informasjonsteknologi, grunnkurs

Eksamensdato: 12. august 2022
Eksamenstid (fra-til): 09:00 – 13:00
Hjelpemiddelkode/Tillatte hjelpemidler: A / Alle hjelpemidler tillatt
Faglige kontakter under eksamen:

- Børge Haugset (tlf.: 934 20 190, kan også kontaktes via Teams)
- Guttorm Sindre (tlf: 94430245, kan også kontaktes via Teams)

Teknisk hjelp under eksamen: NTNU Orakel **Tlf:** 73 59 16 00

Får du tekniske problemer underveis i eksamen, må du ta kontakt for teknisk hjelp snarest mulig, og senest innen eksamenstida løper ut/prøven stenger. Kommer du ikke gjennom umiddelbart, hold linja til du får svar.

ANNEN INFORMASJON:
Ikke ha Inspera åpen i flere faner, eller vær pålogget på flere enheter, samtidig, da dette kan medføre feil med lagring/levering av besvarelsen din.

Skaff deg overblikk over oppgavesettet før du begynner på besvarelsen din.

Les oppgavene nøye, gjør dine egne antagelser og presiser i besvarelsen hvilke forutsetninger du har lagt til grunn i tolkning/avgrensing av oppgaven. Faglig kontaktperson kan kontaktes dersom du mener det er feil eller mangler i oppgavesettet.

Det er angitt i prosent hvor mye hver deloppgave i eksamenssettet teller ved sensur. Hele oppgavesettet gir totalt **100** poeng som tilsvarer 100%. Les gjennom hele oppgavesettet før du begynner å løse oppgaven. Disponer tiden godt!

Gjør dine egne antagelser og presiser i besvarelsen hvilke forutsetninger du har lagt til grunn i tolkning/avgrensing av oppgaven. Faglig kontaktperson skal kun kontaktes dersom det er direkte feil eller mangler i oppgavesettet.

ANNEN VIKTIG INFORMASJON
Lagring: Besvarelsen din i Inspera Assessment lagres automatisk. Jobber du i andre programmer – husk å lagre underveis.

Juks/plagiat: Eksamen skal være et individuelt, selvstendig arbeid. Det er tillatt å bruke hjelpemidler, men vær obs på at du må følge eventuelle anvisningen om kildehenvisninger under. Under eksamen er det ikke tillatt å kommunisere med andre personer om oppgaven eller å distribuere utkast til svar. Slik kommunikasjon er å anse som juks.
Alle besvarelser blir kontrollert for plagiat. *Du kan lese mer om juks og plagiering på eksamen her*.

Kildehenvisninger: Det skal henvises til kilder som brukes *utover* lærebok og slides brukt i faget. Kilder oppgis ved referanse til en bok, eller nettsiden der du fant informasjonen., som "https://no.wikipedia.org/wiki/Binært_tallsystem". Hvis du bruker kode du finner på en nettside, lim inn addressen til denne nettsiden.

Varslinger: Hvis det oppstår behov for å gi beskjeder til kandidatene underveis i eksamen (f.eks. ved feil i oppgavesettet), vil dette bli gjort via varslinger i Inspera. Et varsel vil dukke opp som en dialogboks på skjermen i Inspera. Du kan finne igjen varselet ved å klikke på bjella øverst i høyre hjørne på skjermen. Det vil i tillegg bli sendt SMS til alle kandidater for å sikre at ingen går glipp av viktig informasjon. Ha mobiltelefonen din tilgjengelig.

OM LEVERING:

Automatisk innlevering: Besvarelsen din leveres automatisk når eksamenstida er ute og prøven stenger, forutsatt at minst én oppgave er besvart. Dette skjer selv om du ikke har klikket «Lever og gå tilbake til Dashboard» på siste side i oppgavesettet. Du kan gjenåpne og redigere besvarelsen din så lenge prøven er åpen. Dersom ingen oppgaver er besvart ved prøveslutt, blir ikke besvarelsen din levert. Dette vil anses som “ikke møtt” til eksamen.

Trekk/avbrutt eksamen: Blir du syk under eksamen, eller av andre grunner ønsker å levere blankt/avbryte eksamen, gå til “hamburgermenyen” i øvre høyre hjørne og velg «Lever blankt». Dette kan ikke angres selv om prøven fremdeles er åpen.

Tilgang til besvarelse: Du finner besvarelsen din i Arkiv etter at sluttida for eksamen er passert.

Eksamensstruktur og råd

Eksamen er delt i tre:

- Del 1: teoridel som teller **25%** av den totale vurderingen.
- Del 2: ulike små autorettede programmeringsoppgaver 2a-2h med dra og slipp og innfylling av ting som mangler. Disse teller til sammen **48**%.
- Del 3: denne består av et sett med oppgaver der du selv må skrive koden. Disse teller til sammen **27**%.

Hvordan skal du best programmere på ITGK-eksamen?

1. Du kan godt skrive kode rett inn i oppgavene, hvis du ønsker. Du vil derimot ikke alltid kunne få kjørt koden din. Vi anbefaler ikke dette - når du tross alt har mulighet bør man bruke en ordentlig editor.
2. Du kan se på oppgaven i Inspera, og så løse oppgavene i en ekstern editor. Du vil da kunne kjøre koden din, for å teste om den er korrekt etc. Du må da skrive testene dine selv.
3. Hvis du jobber utenfor Inspera, så kan det være lurt å kopiere inn løsningene etter hvert. Skulle noe skje med datamaskinen din, da har du levert så mye som mulig.

Bruk av eksterne ressurser, og forbud mot samarbeid

1. Husk at du har alle hjelpemidler til rådighet. Dette betyr alle som **ikke** involverer samarbeid med andre personer, uansett om de tar eksamen eller ikke. Hvis du finner kode som passer på en nettside, **skal du** legge nettsiden inn som en kommentar i koden. Alle som programmerer søker opp masse ting på nettet.
2. Vi anbefaler sterkt at dere ikke forsøker dere på samarbeid. Vi kommer til å kjøre en solid sjekk av likheter i både teori og programmeringsoppgaver. Det hjelper ikke å endre variabelnavn og slikt - vi kan finne det. Vi kommer til å lete, men dette er en jobb vi håper vi gjør forgjeves. Derfor: hvis dere kopierer noe fra nettet, og noen andre også har gjort det: det er helt ok, men hvis vi har en referanse slipper vi å anklage dere unødig.

Lykke til!

1 Regler og samtykker

REGLER OG SAMTYKKER

Dette er en **individuell** eksamen. Du har ikke lov til å kommunisere (gjennom web-forum, chat, telefon, hverken i skriftlig, muntlig eller annen form), ei heller samarbeide med noen andre under eksamen.

Før du kan fortsette til selve eksamen må du forstå og SAMTYKKE i følgende:

NTNU-policy for juks ved eksamen:
(**Norsk**) https://innsida.ntnu.no/wiki/-/wiki/Norsk/Juks+på eksamen

Under Eksamen:

Jeg skal IKKE motta hjelp fra andre.

- Aksepter

Jeg skal IKKE hjelpe andre eller dele løsningen min med noen.

- Aksepter

Jeg skal IKKE kopiere noe kode fra noen eksisterende online/offline kilder uten kildehenvisning. Du kan se, og deretter skrive din EGEN versjon av koden, eller legge kilden til i en kommentar.

- Aksepter

Jeg er klar over at jeg kan stryke uavhengig av hvor korrekte svarene mine er, hvis jeg ikke følger reglene og/eller IKKE aksepterer disse utsagnene.

- Aksepter

Jeg er også klar over at juks kan ha alvorlige konsekvenser, som å bli utestengt fra universiteter og få annullert eksamensresultater.

- Aksepter

I tillegg: Jeg gir samtykke til at mine eksamensdetaljer (anonymisert) kan brukes til forskningsformål av ansatte ved NTNU for å forbedre undervisningen i faget.

- ○ Ikke bruk resultatene mine til forskning
- ○ Aksepter

Maks poeng: 0

2 (1a) Transistor (5%)

Forklar kort hvilke fordeler transistoren hadde som komponent i datamaskiner sammenlignet med tidligere teknologier som reléer og vakuumrør.

Skriv ditt svar her (inntil 150 ord)

Format | B I U x_2 x^2 I_x | Ω | Σ

Words: 0

Maks poeng: 5

3 (1b) Binærtall, toerkomplement (5%)

Forklar ved hjelp av eksempler hvordan det binære tallsystemet fungerer for representasjon av heltall med toerkomplement. Som eksempel skal du bruke **20 + siste siffer i ditt kandidatnummer både som positivt og negativt tall.** F.eks., hvis siste siffer i ditt kandidatnummer er 2 (slik som i 10002, 10012, ...), skal du altså bruke tallene **+22** og **-22** som eksempel i forklaringen.

Skriv ditt svar her (inntil 250 ord)

Format

Words: 0

Maks poeng: 5

4 (1c) Representasjon av tekst (5%)

UNICODE-standarden var en forbedring relativt til ASCII og Extended ASCII når det gjaldt representasjon av tekst. På tross av disse forbedringene kan man se følgende i Python:

>>> 'A' < 'B'
True
>>> 'Æ' < 'Å'
False
>>> 'Ø' < 'Å'
False

Som eksemplet viser, blir 'A' korrekt evaluert til å være mindre enn 'B' fordi den kommer tidligere i alfabetet. Tilsvarende skulle 'Æ' og 'Ø' da ha vært mindre enn 'Å', men det er ikke tilfelle.

Forklar i denne oppgaven: (a) på hvilke måter UNICODE var en forbedring sammenlignet med ASCII og Extended ASCII. Og (b) hvorfor evalueres ikke Æ og Ø som mindre enn Å?

Skriv ditt svar her (Inntil 200 ord)

Format

Words: 0

Maks poeng: 5

5 (1d) CPU, klokkefrekvens (5%)

(a) Med tanke på CPU og klokkefrekvens, forklar betydningen av begrepene *overklokking* og *underklokking*. Hva er fordeler og ulemper med overklokking og underklokking?

(b) Firmaet Raske Beregninger AS utfører beregninger hvor det er viktig å få svarene raskt, basert på data som kunder laster opp på firmaets server. Ofte gjør imidlertid tregheter i nettverket at kunder må vente for lenge på svarene. Firmaet har foreløpig ikke tatt i bruk overklokking i maskinen som utfører beregningene. Diskuter hvorvidt dette vil være en anbefalt løsning for at kundene skal få resultatene raskere.

Skriv ditt svar her (inntil 250 ord)

Format

Words: 0

Maks poeng: 5

6 (1e) Sikkerhet (5%)

En morgen merker firmaet Raske Beregninger at ting går enda mye tregere enn vanlig på serveren hvor kunder kan laste opp data for beregninger. De finner at serveren mottar synkroniseringsforespørsler fra en rekke forskjellige datamaskiner både i Norge og utlandet, men at disse maskinene ikke responderer med noen kvittering når forespørselen blir besvart. Hva slags type sikkerhetsangrep virker det sannsynlig at Raske Beregninger er blitt utsatt for her? Forklar litt mer om denne typen sikkerhetsangrep, hvordan de utføres, og hva som er hensikten med dem.

Skriv ditt svar her (inntil 250 ord)

Format

Words: 0

Maks poeng: 5

7 (2a) Fartsbot (4%)

Funksjonen **speeding_fine(f)** skal returnere størrelsen på fartsbot avhengig av bilens fart **f**. Reglene er som følger: Hvis farten er høyere enn 130 blir bota 10000, hvis den er er over 110 (men ikke over 130) blir bota 5000, og hvis den er over 100 (men ikke over 110) blir bota 3000. Hvis farten ikke er over 100, blir det ikke bot i det hele tatt. Eksempler på bruk av funksjonen:

```
>>> speeding_fine(111)
5000
>>> speeding_fine(133)
10000
>>> speeding_fine(110)
3000
>>> speeding_fine(100)
0
```

Dra og slipp kodelinjer til riktig sted under slik at funksjonen virker som den skal.

Hjelp

fart > 130:	bot = 0	bot = 5000	bot = 10000	bot = 3000
fart > 100:	fart > 110:			

```
def speeding_fine(f):
    if [          ]
            [          ]
    elif [          ]
            [          ]
    elif [          ]
            [          ]
    else:
            [          ]
    return bot
```

Maks poeng: 4

8 (2b) Vocals / Vokaler (4%)

Funksjonen **har_vokaler(s)** får inn en streng **s** som kan antas å alltid bestå av bare små bokstaver. Funksjonen skal returnere **True** hvis denne strengen inneholder minst én vokal, **False** hvis den ikke inneholder noen vokal. Eksempel på bruk av funksjonen:

```
>>> har_vokaler('rar')
True
>>> har_vokaler('txt')
False
```

Dra og slipp kodelinjer til riktig posisjon så funksjonen virker som den skal. Hjelp

VOKALER = 'aeiouyæøå'	return True	return False
if b in VOKALER:	for b in s:	

```
def har_vokaler(s):
    
    
        
        
            
```

Maks poeng: 4

9 (2c) Tverrsum (5%)

Tverrsummen til et tall er summen av sifrene i tallet. Tverrsummen til 234 er 9 (2+3+4), og tverrsummen til 5150 er 11 (5+1+5+0). Nedenfor skal det lages tre ulike funksjoner, **cross_sum_1(num)**, **cross_sum_2(num)** og **cross_sum_3(num)** som får inn et positivt heltall num som parameter, og som returnerer tverrsummen til dette heltallet. Dvs., de tre funksjonene løser samme problem, men på tre ulike måter.

Dra og slipp kodelinjer til rett sted slik at alle de tre funksjonene virker som de skal.

Hjelp

for s in set(num):	liste = []	result += num % 10
liste.append(int(s))	while num > 0:	num = str(num)
num = num // 10	svar += int(s) * num.count(s)	result = 0
for s in str(num):		

```
def cross_sum_1(num):
        [ ]
        [ ]
            [ ]
            [ ]
    return result

def cross_sum_2(num):
        [ ]
        [ ]
            [ ]
    return sum(liste)

def cross_sum_3(num):
    svar = 0
        [ ]
        [ ]
            [ ]
    return svar
```

Maks poeng: 5

10 (2d) Telle like tegn (5%)

Funksjonen **count_same(streng, i)** får inn som parametre en **streng** og et heltall **i**. Funksjonen skal ta utgangspunkt i tegnet som befinner seg på strengens indeksposisjon **i**, og returnere som heltall hvor mange like tegn som kommer sammenhengende etter hverandre derfra og videre i strengen (inkludert tegnet man startet på).

Eksempel på bruk av funksjonen:
```
>>> count_same('aabbbcb', 2)
3
>>> count_same('02221333', 0)
1
>>> count_same('0xxxx8888', 5)
4
```

Det første kallet gir 3 som resultat fordi indeksposisjon 2 er den første b'en i strengen, og det herfra står 3 b'er sammenhengende. Det andre kallet gir 1 fordi indeksposisjon 0 har tegnet '0' og det bare er ett sammenhengende av dette. Det siste kallet gir 4 fordi indeksposisjon 5 er til den første 8'eren, hvor det er fire sammenhengende.

Fyll inn riktig innhold i tekstfeltene slik at funksjonen virker som den skal.

```
def count_same(num, i):
    res = [          ]
    n = [          ]
    while i [          ] len(num) [          ] num[i] == [          ]:
        res [          ] 1
        i [          ] 1
    return res
```

Maks poeng: 5

11 (2e) Fjerne like siffer / Remove same digits (9%)

Funksjonen **remove_same_digits(num)** tar inn et heltall som parameter og skal også returnere et heltall hvor påfølgende like sifre fra det opprinnelige heltallet er fjernet på følgende måte:

- hvis det kommer flere enn 0 nuller på rad, skal sifferet 0 fjernes (dvs. 0 skal i praksis alltid fjernes)
- hvis det kommer mer enn 1 ener på rad, skal de sammenhengende enerne fjernes
- hvis det kommer mer enn 2 toere på rad, skal disse sammenhengende toerne fjernes
- Osv., generelt altså at et siffer N skal fjernes hvis det kommer mer enn N N'ere på rad

Eksempel på bruk av funksjonen:

```
>>> remove_same_digits(1223334444)
1223334444
>>> remove_same_digits(10222333)
1333
>>> remove_same_digits(755555533033)
73333
```

I det første eksemplet blir ingenting fjernet fordi vi kan ha inntil en ener, to toere osv. I det andre eksemplet blir 0'en fjernet, samt de tre sammenhengende toerne siden vi maks kan ha to. I det tredje eksemplet blir 0'en fjernet, samt de sammenhengende femmerne siden vi har seks av disse men maks kan ha fem. Merk at returverdien i siste eksempel ender opp med fire sammenhengende treere. Dette er akseptabelt siden de **ikke** var sammenhengende i tallet som ble gitt inn til funksjonen.

Velg riktig alternativ i hvert av feltene nedenfor slik at funksjonen virker som den skal. Funksjonen **count_same()** er som beskrevet i forrige deloppgave, du kan anta at denne virker selv om du ikke har fått den til.

```
def remove_same_digits(num):
    num = str(num)

    [Velg alternativ] (i = 1, i = 0, i = num)

    while [Velg alternativ] (i < len(num), i < num, i <= num):

        [Velg alternativ] (for, if, while)num[i] in '0123456789':

        ct = count_same( [Velg alternativ] (num, i, '0123456789', str(i), '0123456789', i))

        if ct > [Velg alternativ] (num, num[i], int(num[i])):

            num = [Velg alternativ] (num[:i], num.replace('0', ''), num.replace(num[i], '')) +
```

Velg alternativ (num[i+ct:], num.replace('0', ''), num.replace(num[i], ''))

else:

Velg alternativ (i = ct - 1, i += 1, i += ct)

else:

Velg alternativ (i += 1, i += ct, i = ct - 1)

return int(num).

Maks poeng: 9

12 (2f) Filvasking / Clean file (8%)

Vi har ei tekstfil **land.txt** på følgende format (fila kan ha flere rader, men vi viser her bare de fem første):

Russia 17,098,242 11.0%
Canada 9,984,670 6.1%
China 9,706,961 6.3%
United States 9,372,610 6.1%
Brazil 8,515,767 5.6%

Hver rad inneholder info om ett land, med navn på landet, så areal i kvadratkilometer, og til slutt hvor stor prosentandel av jordas landareal dette utgjør. Det er flere uheldige sider ved dette filformatet:

- komma for hver tre siffer i tallet for areal gjør at disse vil ikke bli korrekt tolket som heltall
- prosenttegnet bakerst gjør at det siste tallet ikke blir tolket riktig som flyttall
- mellomrom er brukt som skilletegn mellom ulike data på hver rad. Men noen land kan også ha mellomrom i navnet (f.eks. 'United States' her) slik at splitting på mellomrom ikke vil virke helt etter hensikten.

Vi ønsker derfor å lage en funksjon **clean_file(filnavn)** som leser ei slik fil og skriver ei ny fil hvor filnavn er tillagt **clean_** (dvs. **clean_land.txt** for dette eksemplet). I denne nye fila skal komma og prosenttegn være fjernet, og semikolon **;** skal være brukt som skilletegn i stedet for mellomrom slik at det ikke blir problem med land som har flere ord i navnet. Med eksemplet over skal resultatfila se ut som følger:

Russia;17098242;11.0
Canada;9984670;6.1
China;9706961;6.3
United States;9372610;6.1
Brazil;8515767;5.6

Velg riktig alternativ i hvert av feltene under slik at funksjonen vil virke som den skal.

```
def clean_file(filnavn):

    with [Velg alternativ] (close, open, clean_file)(filnavn) as infile:
        info = infile.read()

    info = info.[Velg alternativ] (split(), find(',','%'), replace(',', '')).replace('%', '')

    liste = info.[Velg alternativ] (split('\n'), split(), split(','))

    for i in [Velg alternativ] (range(len(liste)), liste, range(liste)):

        liste[i] = liste[i] [Velg alternativ] (remove('%'), .split(), append('%'))
```

```
        liste[i] Velg alternativ ([0:-2], [j], [i+1]) = [' '.join(liste[i] Velg alternativ ([0:-2], [j],
[i+1]))]

        liste[i] = Velg alternativ (' ', '%', ';').join(liste[i])

    clean_info = Velg alternativ ('\n', ' ', ';').join(liste)

    with open('clean_'  + filnavn, Velg alternativ ('a', 'w', 'r')) as outfile:
        outfile.write(clean_info)
```

Maks poeng: 8

13 (2g) Fil til ordbok / File to dictionary (10%)

Anta nå at vi har ei fil på formatet som ble laget i forrige deloppgave, for eksempel

Russia;17098242;11.0
Canada;9984670;6.1
China;9706961;6.3
United States;9372610;6.1
Brazil;8515767;5.6

Vi ønsker en funksjon file_2_dictionary(filnavn) som leser ei slik fil og returnerer ei ordbok (dictionary) hvor navn på landet er nøkkel, og verdi er ei liste hvor fremste element er et heltall for arealet, og bakerste element er et flyttall for hvor stor prosetandel landet har av Jordas totale landareal. Med filinnhold som vist over, skal returnert ordbok bli:

{ 'Russia' : [17098242, 11.0] , 'Canada' : [9984670, 6.1], 'China' : [9706961, 6.3], 'United States' : [9372610, 6.1], 'Brazil' : [8515767, 5.6] }

Fyll inn det som mangler i funksjonen slik at den virker som den skal.

```
def file_2_dictionary(filnavn):
    fil = [          ] (filnavn)
    [      ] = { }
    for [      ] in [          ] :
        data = line.split('[      ]')
        areal[[          ]] = [[          ] , [          ]]
    fil.[      ]
    return [      ]
```

Maks poeng: 5

14 (2h) Unntaksbehandling (8%)

Nedenfor er vist en enkel funksjon som lar bruker skrive inn to opplysninger via tastaturet, ett heltall og ett flyttall:

```
def hent_data_fra_bruker():
    h = int(input("Oppgi høyde i cm (heltall): "))
    v = float(input("Oppgi vekt i kg med en desimal: "))
    return h, v
```

Hvis brukeren skriver noe annet enn tall, eller gir desimaler også på høyden, eller feilaktig bruker komma i stedet for punktum i flyttallet for vekt, vil denne funksjonen feile og gi ValueError. Vi ønsker derfor å lage en ny versjon av funksjonen som bruker unntaksbehandling for å hindre slik feil, og gå i løkke inntil brukeren har gitt inn data som korrekt lar seg konvertere med int() og float(). Eksempel på kjøring av den reviderte versjonen:

```
>>> hent_data_fra_bruker()
Oppgi høyde i cm: 180.5
Høyde må oppgis som et helt tall, f.eks. 179
Vekt må oppgis som et flyttall,  f.eks. 81.2

Oppgi høyde i cm: 181
Oppgi vekt med en desimal, kg.g: 82,7
Høyde må oppgis som et helt tall, f.eks. 179
Vekt må oppgis som et flyttall,  f.eks. 81.2

Oppgi høyde i cm: 181
Oppgi vekt med en desimal, kg.g: 82.7
(181, 82.7)
```

Dra og slipp kodelinjer til riktig posisjon slik at funksjonen virker som den skal.

Hjelp

- while not ok:
- v = float(input("Oppgi vekt i kg med en desimal: "))
- print("Vekt må oppgis som et flyttall, f.eks. 81.2\n")
- h = int(input("Oppgi høyde i cm: "))
- ok = False
- ok = True
- print("Høyde må oppgis som et helt tall, f.eks. 179")
- try:
- except:

```
def hent_data_fra_bruker():
                    
                    
                        
                            
                            
                            
                        
                        
                    
                    
                    
                        
                        
    return h, v
```

Maks poeng: 8

15 (3a) Summere tall (5%)

Funksjonen **sum_pos_odd()** skal ta inn ei liste med heltall og returnere summen av de positive oddetallene i lista. Dvs., partall skal ikke tas med i summen, og negative tall skal heller ikke tas med.

Eksempel på bruk av funksjonen:

>>> sum_pos_odd([2,-3,5,-1,7,6,4])
12

Svaret blir her 12 fordi de eneste positive oddetallene i lista er 5 og 7, og 5+7 blir 12.

Skriv koden for funksjonen sum_pos_odd()

```
1
```

Maks poeng: 5

16 (3b) Lengste streak / Longest streak (5%)

I en del typer sport er det slik at seier gir 3 poeng, uavgjort 1 poeng og tap 0 poeng. Anta at vi har prestasjonen til et lag gjennom en sesong i form av en streng, f.eks.
'000111101333313003311111'
Vi ønsker en funksjon longest_streak(streng, resultat) som får inn som parameter en slik streng, samt hva slags resultat vi ønsker å finne streak for (3 for seiers-streak, 1 for uavgjort-streak, 0 for taps-streak). Funksjonen skal returnere den lengste streaken som kan finnes for det aktuelle resultatet. Eksempler på kjøring av funksjonen:

```
>>> longest_streak( '000111101333313003311111', 3 )
4
>>> longest_streak( '000111101333313003311111', 1 )
5
>>> longest_streak( '000111101333313003311111', 0 )
3
```

Det første kallet gir 4 som resultat, fordi lengste streak av 3-tall (andre argument) er 4, om trent midt i strengen. Det andre kallet gir 5, fordi dette er lengste streak av 1-tall, helt sist i strengen. Det tredje kallet gir 3 som resultat, fordi dette er lengste streak av 0, helt fremst i strengen. Funksjonen din må virke for generelle strenger av denne typen, som kan inneholde en vilkårlig lang sekvens av sifrene 0, 1 og 3 i vilkårlig rekkefølge.

Skriv koden for funksjonen longest_streak() her:

```
1
```

Maks poeng: 5

17 (3c) Fra 2D-liste til ordbok (7%)

Vi har gitt ei 2D liste (liste av lister) hvor hver rad inneholder opplysninger om en person, nemlig navn og et 8-sifret telefonnummer samt gps-koordinater i form av breddegrad og lengdegrad for hvor vedkommende befant seg på et visst klokkeslett. Eksempel på mulig innhold:

L = [['Anna', 12131415, 41.40309, 2.15203],
['Nils', 21222222, 63.40309, 5.19909],
['Don', 21558888, 63.40309, 5.19909],
['Lea', 77131415, 41.40309, 2.15203],
['Joe', 99131415, 25.22232, 19.33213]]

NB: Lista kan ha mange flere rader enn dette, som bare er for å vise eksempel på innhold og hvordan formatet er. Koden din må virke for vilkårlig lange lister av denne typen.

Funksjonen co_location(L) skal ta inn ei slik 2D-liste og returnere ei ordbok (en dictionary) hvor et tuppel basert på GPS-koordinatene skal være nøkkel, og hvor dermed personer som befant seg på samme sted skal være i verdilista tilknyttet denne nøkkelverdien. Med eksemplet over skal resulterende ordbok bli:

{ (41.40309, 2.15203): [['Anna', 12131415], ['Lea', 77131415]],
(63.40309, 5.19909): [['Nils', 21222222], ['Don', 21558888]],
(25.22232, 19.33213): [['Joe', 99131415]] }

Som man ser, har Anna og Lea havnet på samme nøkkel i ordboka fordi de hadde samme GPS-koordinater, og likeledes Nils og Don. For hver GPS-koordinat skal verdidelen til høyre for kolon fremdeles være ei 2D-liste, men hvor hver indre liste nå kun har navn og telefonnummer, siden koordinatene er i nøkkelen.

Skriv koden for funksjonen co_location(L)

```
1
```

Maks poeng: 7

18 (3d) Finne nærliggende personer (10%)

En person er savnet og vi ønsker å finne vitner som kan ha sett personen ved å sjekke hvem som befant seg på stedet vedkommende sist var observert. Du kan anta at du har tilgang til ei liste L på samme format som i forrige deloppgave, altså:

L = [['Anna', 12131415, 41.40309, 2.15203],
['Nils', 21222222, 63.40309, 5.19909],
['Don', 21558888, 63.40309, 5.19909],
['Lea', 77131415, 41.40309, 2.15203],
['Joe', 99131415, 25.22232, 19.33213]]

I denne oppgaven skal du lage et skript som:

- lar brukeren skrive inn fra tastaturet den savnede personens navn (NB: dette behøver ikke være en person som står i lista L, dvs. kan være en helt ny person)
- også lar brukeren skrive inn fra tastaturet koordinater for breddegrad og lengdegrad for siste observasjon av personen
- responderer med å liste opp posisjon, navn og telefonnummer for eventuelle personer som har vært tilstrekkelig nærme, her definert som et avvik på mindre eller lik 0.00003 på hver av koordinatene.

Eksempel på to kjøring av skriptet, hvor den første ikke finner noen nærliggende personer, mens den andre finner to som var tilstrekkelig nær:

>>> %Run coloc.py
Name of missing person: Galadriel
Latitude of last observation: 43.44444
Longitude of last observation: 2.33333
No nearby persons
>>> %Run coloc.py
Name of missing person: Frodo
Latitude of last observation: 41.40311
Longitude of last observation: 2.15201
Nearby persons:
(41.40309, 2.15203): Anna (12131415) ; Lea (77131415) ;

Du står fritt til å kalle funksjonen co_location() fra forrige deloppgave hvis du ønsker det, og kan anta at denne virker som forutsatt selv om du ikke har fått den til. Det er også lov å dele opp koden i flere hjelpefunksjoner, du trenger ikke skrive hele som et skript.

Skriv koden din her

```
1
```

Maks poeng: 10