Løsningsforslag for konteksamen i TDT4109/ TDT4110 – Informasjonsteknologi, grunnkurs 12. august 2020

Oppgave 1.1
Forklar kort hvorfor har datamaskiner ofte et minnehierarki bestående av registre, cache, primærminne (RAM), og sekundærminne.

Løsning:
Minnehierarki er organisert i henhold til hastighet der registre er kjappest, så cache, så primærminne og til slutt sekundærminne som er tregest. Grunnen til at man f.eks. ikke bruker bare det raskeste minnet er relatert til pris (raskere minne er dyrere per byte enn tregere minne), det er fysiske begrensninger hvor mange registre man kan ha chip, og sekundærminne overlever etter at strømmen til datamaskinen er slått av.

Oppgave 1.2
Forklar kort hvordan analog lyd kan overføres og representeres i en datamaskin:

Løsning:
Lyd i naturen består av lydbølger som har en amplitude (høyde på bølgen) og en frekvens (hastighet på bølgen). For å kunne lage en representasjon som datamaskinen kan håndtere for lyd, må lyden først samples ved hjelp av en analog-til-digital konverter (ADC). Dette vil si at man måler på gitte tidspunkt amplituden på lydbølgen og representerer hvor høy bølgen (både positivt og negativt) med et antall bit. Hvis man f.eks. sampler med 8-bit kan man totalt representere 256 ulike verdier av amplituden av lydbølgen. Hvis man sampler med 16-bit kan man totalt representere 65536 ulike verdier som gir 2^{15} (32768) variasjoner for positive verdier og 32668 for negative verdier. I tillegg til avlesning av amplituden, er det viktig at man sampler lydbølgen i høy nok frekvens slik at man klarer å fange opp og representere svingningene i lydbølgen godt nok. Dette kalles samplingsfrekvens.

Oppgave 1.3
Forklar kort hvorfor tekstkodingen UTF-8 er bedre egnet for koding av tekst av websider enn det ASCII er:

Løsning:
ASCII-formatet er begrenset i antall tegn det kan representere (maks 255 ved 8bit) ettersom hvert tegn i formatet representeres med 1 byte. Dette gjør det ikke mulig å representere alle type skriftspråk med ulike symboler og tegn, som ikke gjør ASCII velegnet for representasjon av websider (som skal fungere for hele verden). Derimot for UTF-8 kan fungerer halvparten av kodingen på samme måte som for ASCII (der første bitet er 0). Derimot hvis første bitet er 1, så kan man representere karakterer med 2, 3 eller 4 bytes. Dette gjør det mulig å kode alle skriftspråk som gjør UTF-8 egnet for koding av websider.

Oppgave 1.4

Forklar kort forskjellen på destruktiv (lossy) og tapsløs (lossless) kompresjon og gi eksempel på bruksområdene for begge:

Løsning:
Tapsløs kompresjon går ut på at man komprimerer data ved å representere lengre sekvenser av 0ere eller 1ere på en kompakt måte (spesifiserer først antall like tall, så om det er en 1 eller 0). Denne type komprimering fungerer best på datarepresentasjoner som har store deler som består av sekvenser av like tall. F.eks. hvis man har et bilde, der mange punkter etter hverandre har samme farge. Fordelen med tapsløs kompresjon er at man ikke mister noe data når man komprimerer. Ulempen er at der kan være mer begrenset hvor mye man klarer å komprimere og at det ikke fungerer hvis man har data der man ikke har sekvenser av 0er eller 1ere.
Destruktiv koding f.eks. av bilder gjør endringer i bildet på små forskjeller f.eks. i fargenyanser som ikke er så lett for et menneskelig øye å oppdage og endrer farger som er ganske like til å bli helt like. Ved destruktiv koding kan man ofte bestemme i hvor stor grad man skal gjøre endringer på f.eks. et bilde. Jo større endringer (forenklinger i representasjon) man tillater, jo mere komprimert blir data-representasjon. En ulempe med destruktiv kode er at man ikke kan få tilbake full representasjon ved å de-komprimere. Komprimeringen av dataen gjør faktiske endringer på data-representasjon. En fordel med denne type koding er at man kan komprimere data langt mer enn tapsløs og at man kan også f.eks. tilpasse kompresjon dynamisk relatert til båndbredde ved streaming av video.

Oppgave 1.5
Forklar kort forskjellen på linjesvitsjing (circuit switching) og pakkesvitsjing (packet switching) i nettverk:

Løsning:
Linjesvitsjing går ut på at man endrer på kablingen fra et punkt til et annet punkt i nettet, slik at overføring mellom punktene kan foregå direkte for en gitt koplet rute. Linjesvitsjing er den gamle måten å knytte punkter sammen på og fungerer på samme måte som gamle telefonsentraler der man koblet personer direkte sammen på en linje.
Pakkesvitsjing går ut på at man deler opp data som skal overføres i flere pakker som sender igjennom nettet. Disse pakkene kan sendes i ulike ruter og kan også komme fram på ulike tidspunkt, men blir satt sammen i riktig rekkefølge hos mottaker. Man har også mekanismer for å sende pakker på nytt hvis noen har kommet bort på veien. Pakkesvitsjing er det som brukes på internett i dag.

Oppgave 1.6
Et firma blir stadigvekk angrepet med *replay*. Hvilke sikkerhetsteknologier bør firma bruke mot et slikt angrep?

Løsning:
Mot replay, er det viktig at all data som sendes er *kryptert* slik at det ikke er enkelt å finne ut hva som er innloggingsinformasjon. I tillegg kan man benytte seg av *hashing*, slik at hvis man kombinerer en pakke som er stjålet med annen data, så vil det avsløres at dataintegriteten ikke er i orden. Man kan også kreve *digitale signaturer* som sikrer riktig avsender på data som sendes. *Brannmurer* kan også brukes slik at man gir tilgang til kun datamaskiner med gitte IP-adresser. Andre løsninger kan være å ha et *intrusion detection system* som kan

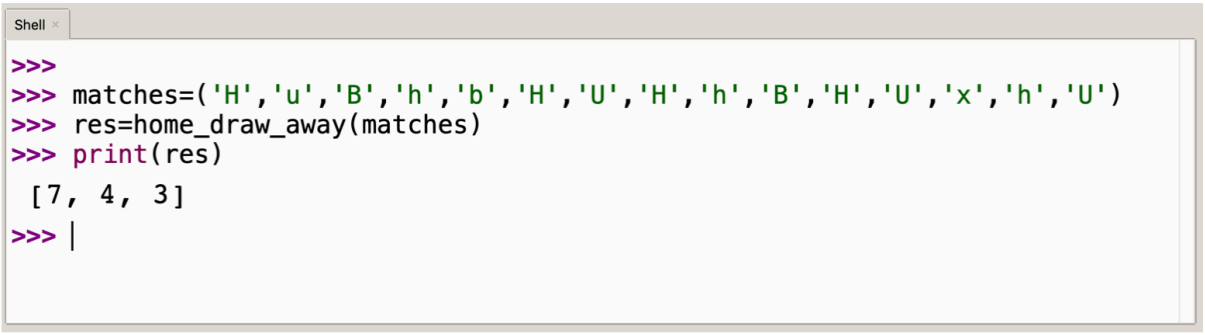
detektere unormal nett-trafikk. Man kan også benytte c*ontent scanning & deep packet inspection* som undersøker data som sendes og leter etter mistenkelige signaturer.

Oppgave 2.1

Lag funksjonen ***home_draw_away*** som har en input-parameter ***matches*** som er en liste som inneholder informasjon om fotballkamper i siste runde ble hjemmeseier ("H"), uavgjort ("U") eller borteseier ("B").

Funksjonen skal returnere ei liste med 3 tall, der det første tallet er antallet hjemmeseire, det andre tallet er antallet uavgjort, og det siste tallet er antallet borteseiere. Funksjonen skal fungere på lister av ulik lengde, skal fungere for stor og liten bokstav for H, U og B, samt at den skal ignorere resultat som ikke er H, U, eller B.

Eksempel på kjøring av kode:



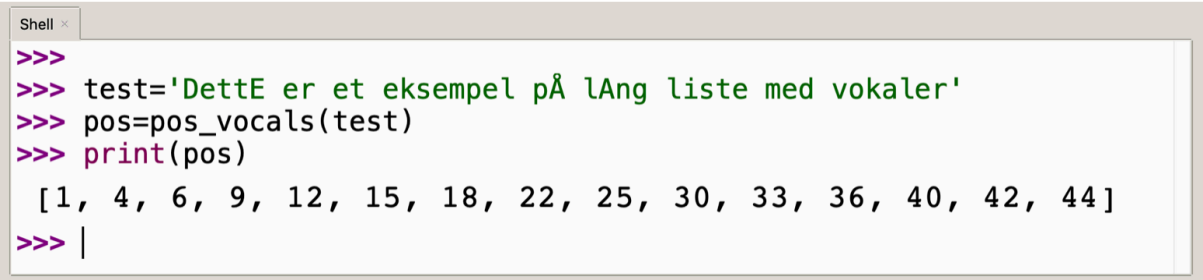
Løsning:

```
def home_draw_away(matches):
    result = [0,0,0]
    for el in matches:
        if el.lower() == 'h':
            result[0] += 1
        elif el.lower() == 'u':
            result[1] += 1
        elif el.lower() == 'b':
            result[2] += 1
    return result
```

Oppgave 2.2

Lag funksjon ***pos_vocals*** som tar inn en parameter ***text*** som er en tekststreng og returnerer posisjonen til alle norske vokaler i strengen ***text*** som ei liste. Funksjonen skal fungere for vokaler skrevet med stor og liten bokstav.

Eksempel på kjøring av kode:



Løsning:

```
def pos_vocals(text):
    vokal = 'aeiouyæøå'
    text = text.lower()
    posisjon = []
    i = 0
    for bokstav in text:
        if bokstav in vokal:
            posisjon.append(i)
        i += 1
    return posisjon
```

Oppgave 2.3

Lag funksjon ***uniqe*** som har en inn-parameter ***text*** som er en tekststreng.

Funksjonen skal returnere en liste med alle unike ord i teksten skrevet med små bokstaver. Rekkefølgen ordene i lista har er ikke bestemt. Merk at funksjonen skal returnere kun ord og tegn som man typisk har i en setning skal fjernes. Man kan anta at tekstrengen starter med en bokstav, at ordene er adskilt med mellomrom, at man ikke har tegn inne i et ord, og at man ikke har flere enn et tegn etter hverandre.

Eksempel på kjøring av kode:

Løsning:

```
def unique(test):
    liste = test.lower().split() # lager liste med ord
    tegn = '.,:;?!-_*@+'         # definerer tegn som skal fjernes
    unikeOrd = []                # nullstiller liste
    for i in range(len(liste)):
        if liste[i][-1] in tegn:  # Hvis siste tegn i et ord er ulovlig
            liste[i] = liste[i][:-1]  # fjern det
        if liste[i] not in unikeOrd:
            unikeOrd.append(liste[i])
    return unikeOrd
```

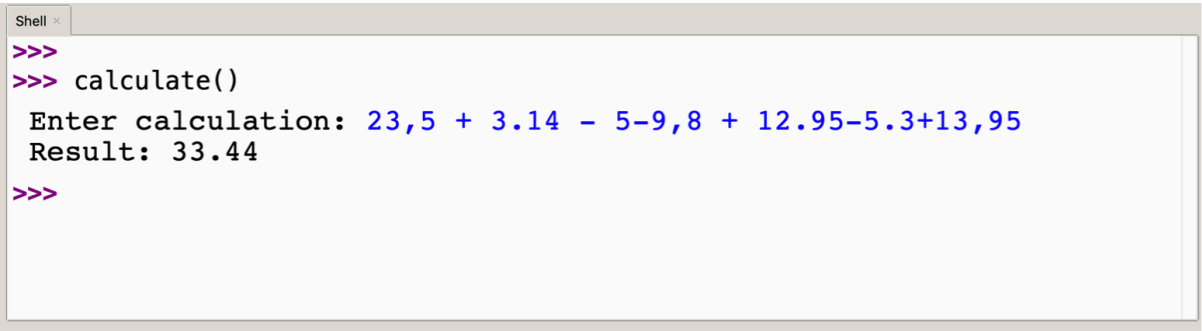
Oppgave 2.4

Lag funksjonen ***calculate*** som har ingen inn-parametere, men som skal be brukeren om å skrive inn et regnestykke som kun kan inneholde tall, + og -. Funksjonen skal ta hensyn til at man kan skrive desimaltall både med desimalkomma og desimalpunktum. Det må også tas hensyn til at det kan skrives åpenrom (mellomrom) i uttrykket. Løsningen kan ikke benytte seg av den innebygde funksjonen eval() eller lignende.

Funksjonen skal først spørre brukeren: "Enter calculation" - der brukeren skal skrive inn et regnestykke.

Funksjonen beregne svaret og skrive ut svaret til konsollet etter teksten: "Result: "

Eksempel på kjøring av kode:



Løsning:

```
def calculate():
    comp = input('Enter calculation: ').replace(',','.').replace(' ','')
    minliste = comp.split('+') # splitter først på +
    mellomliste = []
    summ = 0
    for el in minliste: # går gjennom element for element
        if '-' in el:   # hvis tegnet - finnes i element
            delliste = el.split('-')    # splitt på -
            summ += float(delliste[0])  # legg til det første elementet
            for i in range(1,len(delliste)):  #trekk fra de andre
                summ -= float(delliste[i])
        else:
            summ += float(el)
    print(f'Result: {summ:.2f}')
```

Oppgave 3.1

Lag funksjonen ***sec_to_time*** som har en input-parameter **s** som er antall sekunder angitt som et heltall og som skal regne om dette til formatet t:m:s, og returnere resultatet som en tekststreng. Her står t for time, m for minutt og s for sekunder. Hvis tiden er mindre enn 1 time skal den returnere 'm:s'. Ellers skal den returnere 't:m:s'.

Eks. på riktig output: '0:45','2:23', '12:04','1:02:09' etc.

Legg merke til at hvis antall sekunder (og minutter hvis det er over en time) er mindre enn 10 skal det det skrives en 0 først.

Eksempel på kjøring av kode:

Løsning:

```
def sec_to_time(s):
    time = s//3600
    min = str((s-time*3600)//60)
    sek = str((s-time*3600)%60)
    if len(sek) < 2 :
        sek = '0' + sek
    if time > 0:
        if len(min) < 2 :
            min = '0' + min
        return str(time) + ':' + min+':'+sek
    else:
        return min+':'+sek
```

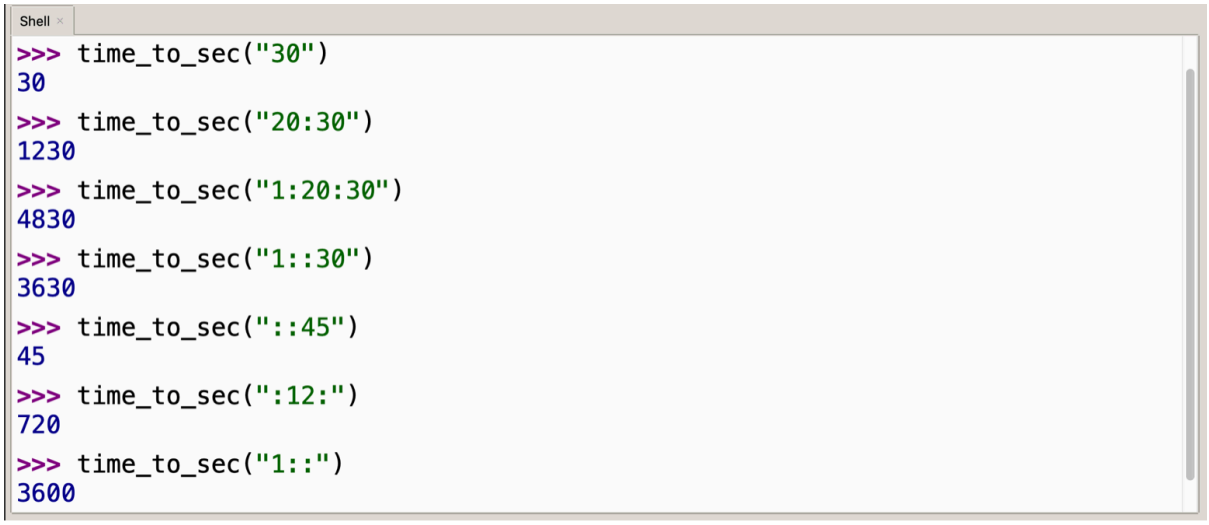
Oppgave 3.2

Lag funksjonen ***time_to_sec*** som tar inn en input-parameter ***tstring*** som er en streng med tidsinformasjon spesifisert som "time:minutt:sekund" og regner den om til antall sekunder. Det er ikke gitt at alle elementene er med. Funksjonen må håndtere strenger som "30","20:30","1:20:30","1::30","::45",":12:","1::".

Funksjonen behøver ikke ta hensyn til strenger som ikke følger dette formatet.

Funksjonen skal returnere antall sekunder strengen representerer som et heltall.

Eksempel på kjøring av kode:



Løsning:

```
def time_to_sec(tstring):
    liste = tstring.split(':')
    sek = 0
    if len(liste) == 1:
        sek = int(liste[0])
    elif len(liste) == 2:
        sek = int(liste[0])*60 + int(liste[1])
    else:
        if liste[0] != '':
            sek = int(liste[0])*3600
        if liste[1] != '':
            sek += int(liste[1])*60
        if liste[2] != '':
            sek += int(liste[2])
    return sek
```

Oppgave 3.3

Lag funksjonen ***enter_song*** som ikke har noe input-parameter, der brukeren kan skrive inn data for en sang med tastaturet. Data som skal skrives inn er Tittel; Artist; Sjanger; Lengde (i min:sek) *med semikolon mellom hvert element*. Funksjonen skal returnere en liste på formatet ['Tittel','Artist','Sjanger','m:s'].

Funksjonen skal først skrive følgende til skjerm: "Enter song (Title;Artist;Genre;m:s)", og deretter en ny linje der brukeren skal skrive inn informasjon med prompt (spørretekst) "> ".

Hvis det skrives inn ugyldige data skal funksjonen skrive teksten "Ugyldige data" og spørre brukeren om å skrive inn data på nytt. Dataene er ugyldig hvis:

- Man ikke har lagt inn 4 verdier (dvs. fire elementer med semikolon mellom).
- Lengden er ikke angitt på riktig tidsformat (m:s) der s er et tall mellom 0 og 59, og m er et tall større enn 0.

Eksempel på kjøring av kode:

```
Shell
>>> enter_song()
 Enter song (Title;Artist;Genre;m:s)
 > Africa;Toto
 Ugyldige data! Formatet må være: Title;Artist;Genre;m:s
 > Africa;Toto;Rock
 Ugyldige data! Formatet må være: Title;Artist;Genre;m:s
 > Africa;Toto;Rock;5:99
 Ugyldige data! Formatet må være: Title;Artist;Genre;m:s
 > Africa;Toto;Rock;5:31
['Africa', 'Toto', 'Rock', '5:31']
>>>
```

Løsning:

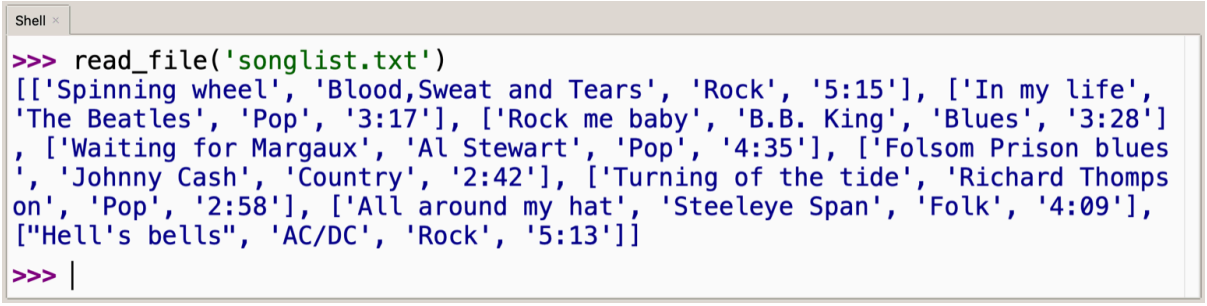
```
def feilmelding():
    print('Ugyldige data! Formatet må være: Title;Artist;Genre;m:s')

def enter_song():
    print('Enter song (Title;Artist;Genre;m:s)')
    while True:
        sang = input('> ')
        sang = sang.split(';')
        if len(sang) != 4:
            feilmelding()
        else:
            if len(sang[-1].split(':')) == 2:
                min = sang[-1].split(':')[0]
                sek = sang[-1].split(':')[1]
                if min.isdigit() and sek.isdigit()\
                   and int(min) < 60 and int(sek) < 60:
                    return sang
                else:
                    feilmelding()
            else:
                feilmelding()
```

Oppgave 3.4

Lag funksjonen ***read_file*** som har en input-parameter ***filename*** som er navnet på filen som skal leses og som returnerer en liste som inneholder en liste med de dataene som finnes i filen. Hver linje i tekstfilen skal representeres som en liste. Funksjonen skal altså returnere en liste med lister. Funksjonen trenger ikke ta hensyn til om filen eksisterer eller problemer med lesing av fil.Anta at tekstfilen *songlist.txt* inneholder følgende:

```
Spinning wheel;Blood,Sweat and Tears;Rock;5:15
In my life;The Beatles;Pop;3:17
Rock me baby;B.B. King;Blues;3:28
Waiting for Margaux;Al Stewart;Pop;4:35
Folsom Prison blues;Johnny Cash;Country;2:42
Turning of the tide;Richard Thompson;Pop;2:58
All around my hat;Steeleye Span;Folk;4:09
Hell's bells;AC/DC;Rock;5:13
```

Eksempel på kjøring av data med *songlist.txt* :



Løsning:

```
def read_file(filename):
    f = open(filename,'r')
    liste = []
    for linje in f:
        liste.append(linje.strip().split(';'))
    f.close()
    return liste
```

Oppgave 3.5

Lag funksjonen ***list_content*** som har to input-parametere: ***filename*** som er en tekststreng som angir navnet på fila som det skal leses fra og ***choice*** som er en tekststreng som spesifiserer hvilken type data som skal listes. Parameteren ***choice*** kan ha tre ulike verdier:

- 'title' - da skal alle *unike* sangtitler i fila ***filename*** returneres som ei liste.
- 'artist' - da skal alle *unike* artister i fila ***filename*** returneres som ei liste.
- 'genre' - da skal alle *unike* sjangere i fila ***filename*** returneres som ei liste.

Merk at det skal gå an å skrive 'title','artist' og 'genre' med store og små bokstaver.

Hvis ikke gyldig ***choice*** er angitt skal tom liste returneres.

Fila som det skal leses av her formatert som vist i Oppgave 3.4. Man trenger ikke ta hensyn til om fil ikke eksisterer eller problemer med å lese fra fil.

Listene skal ikke inneholde duplikater (dvs. flere like elementer) og rekkefølgene på elementene er ikke bestemt.

Løsningen skal benytte seg av funksjoner spesifisert i tidligere oppgaver så langt dette er mulig!

Eksempel på kjøring av kode (ved bruk av fil fra Oppgave 3.4):

```
Shell
>>> list_content('songlist.txt','person')
[]
>>> list_content('songlist.txt','title')
['Spinning wheel', 'All around my hat', 'Waiting for Margaux', 'Rock me baby'
, 'In my life', 'Turning of the tide', 'Folsom Prison blues', "Hell's bells"]
>>> list_content('songlist.txt','artist')
['AC/DC', 'The Beatles', 'Blood,Sweat and Tears', 'Johnny Cash', 'Richard Tho
mpson', 'Steeleye Span', 'Al Stewart', 'B.B. King']
>>> list_content('songlist.txt','genre')
['Country', 'Pop', 'Rock', 'Blues', 'Folk']
>>>
```

Løsning:

```
def list_content(filename,choice):
    liste = []
    if choice.lower() == 'title': indeks = 0
    elif choice.lower() == 'genre': indeks = 2
    elif choice.lower() == 'artist': indeks = 1
    else: return liste
    oversikt = read_file(filename)
    for linje in oversikt:
        if indeks != 0:
            if linje[indeks] not in liste:
                liste.append(linje[indeks])
        else:
            liste.append(linje[indeks])
    return liste
```

Oppgave 3.6

Radioverten har behov for å kunne se en liste av tilgjengelige sanger basert på en sjanger.

Lag funksjonen ***list_songs_genre*** som har en inn-parameter ***filename*** som er av typen tekststreng og spesifiserer navnet på fila data skal leses fra.

Funksjonen skal først spørre brukeren om hvilken sjanger man ønsker å få listet opp sanger fra. Dette gjøres ved at man får oppgitt en liste over tilgjengelige sjangere som finnes i fila ***filename*** og deretter spør brukeren om å skrive inn ønsket sjanger.

Spørsmålet til brukeren skal se slik ut (tilgjengelige sjangere kan variere):

Choose genre [Blues,Country,Rock,Pop]:

Funksjonen skal returnere en liste av alle sangene for valgt sjanger der hver sang er ei liste bestående av tittel, artist og spilletid, samt den totale spilletiden som en tekststreng på formatet t:m:s, der t=time, m=minutt og s=sekunder. Hvis sjanger ikke finnes, skal funksjonen returnere en tom liste og tiden 0:0.

Løsningen skal benytte seg av funksjoner spesifisert i tidligere oppgaver så langt dette er mulig!

Eksempel på kjøring av kode (med tekstfil som spesifisert i Oppgave 3.4):

```
Shell
>>> list_songs_genre('songlist.txt')
 Choose genre [Rock,Folk,Blues,Pop,Country]: Classic
([], '0:0')
>>> list_songs_genre('songlist.txt')
 Choose genre [Rock,Folk,Blues,Pop,Country]: Rock
([['Spinning wheel', 'Blood,Sweat and Tears', '5:15'], ["Hell's bells", 'AC/D
C', '5:13']], '10:28')
>>> list_songs_genre('songlist.txt')
 Choose genre [Rock,Folk,Blues,Pop,Country]: Pop
([['In my life', 'The Beatles', '3:17'], ['Waiting for Margaux', 'Al Stewart'
, '4:35'], ['Turning of the tide', 'Richard Thompson', '2:58']], '10:50')
>>>
```

Løsning:

```
def list_songs_genre(filename):
    sjangre = list_content(filename,'genre')
    sjanger = input(f'Choose genre {sjangre}:')
    sjanger = sjanger[0].upper() + sjanger[1:].lower()
    sangliste = []
    summ=0
    liste = read_file(filename)
    for el in liste:
        if el[2] == sjanger:
            sangliste.append([el[0],el[1],el[3]])
            summ += time_to_sec(el[3])
    return sangliste,sec_to_time(summ)
```

Oppgave 3.7

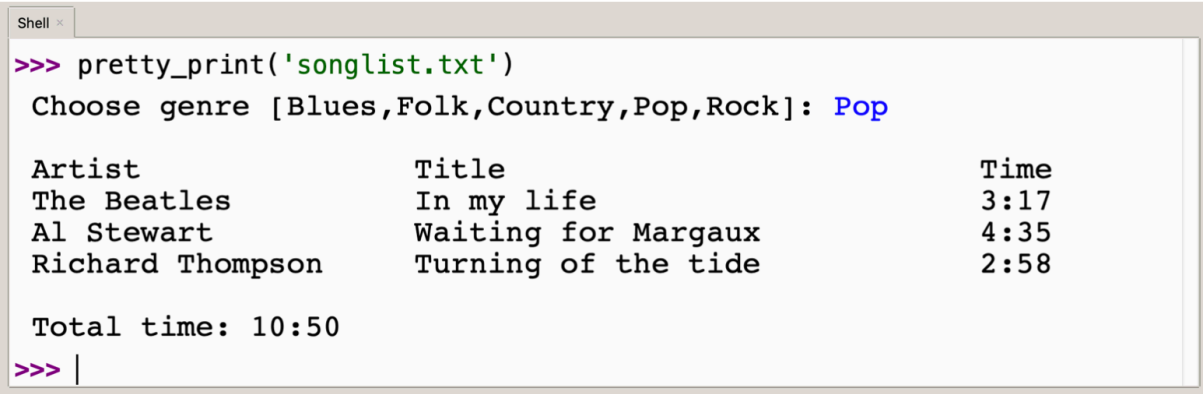
Skriv funksjonen ***pretty_print*** som har en inn-parameter ***filename*** som er en tekststreng som spesifiserer tekstfila som data skal leses fra. Funksjonen *skal bruke* funksjonen ***list_songs_genre*** fra Oppgave 3.6 til å hente inn sanger for en spesifisert sjanger og skrive dem pent ut på formatet: Artist (20 tegn venstrejustert), Title (30 tegn venstrejustert) og Time.

Merk at hvis navnet på artist er lengre enn 20 tegn, skal kun de 20 første tegn skrives ut, og at hvis tittel på sangen er over 30 tegn, så skal kun de 30 første tegn skrives ut.

Funksjonen skal skrive ut "Total time:" som er total spilletid for sangene som er listet opp.

Man trenger ikke å ta hensyn til feil filnavn eller problemer med lesing av fil.

Eksempel på kjøring av kode med data fra tekstfila songlist.txt (spesifisert i Oppgave 3.4):



Løsning:

```
def pretty_print(filename):
    sanger,tid = list_songs_genre(filename)
    print(f'\n{"Artist":<20}{"Title":<30}{"Time"}')
    for el in sanger:
        print(f'{el[1][0:19]:<20}{el[0][0:29]:<30}{el[2]}')
    print(f'\nTotal tid:{tid:>10}')
```

Oppgave 3.8

Skriv funksjonen ***add_song_to_file*** som har en inn-parameter ***filename*** som er av typen tekststreng som er navnet på tekstfila som skal oppdateres med nye sanger.

Brukeren skal bli spurt om å legge inn informasjon om en ny sang ved hjelp av tastaturet på formatet:

Enter song (Title;Artist;Genre;m:s):

Man skal sjekke at input er korrekt iht. til antall elementer og tidsangivelse. Funksjonen skal legge sangdata som brukeren har skrevet inn bakerst i tekstfila ***filename***. Data som allerede er registrert skal ikke slettes! Funksjonen skal sjekke *at sangen ikke er registrert fra før* og må da sjekke både sangtittel og artist, da samme sang kan vÃ¦re innspilt av flere artister. Hvis sangen er registrert fra før, skal funksjonen skrive ut: "*[tittel på sangen] already exist in [**filename**]*" (se eksempel fra kjøring under).
Løsningen skal bruke funksjoner fra tidligere oppgaver hvis dette er mulig!

Eksempel på kjøring av kode med fila *songlist.txt* fra Oppgave 3d:

```
Shell
>>> add_song_to_file('songlist.txt')
 Enter song (Title;Artist;Genre;m:s)
 > In my life;The Beatles;Pop;3:17
 In my life already exists in songlist.txt
>>> add_song_to_file('songlist.txt')
 Enter song (Title;Artist;Genre;m:s)
 > In my life;Joseph Vincent;Pop;1:46
 In my life has been added to songlist.txt
>>> add_song_to_file('songlist.txt')
 Enter song (Title;Artist;Genre;m:s)
 > Africa;Toto;Pop;4:56
 Africa has been added to songlist.txt
>>>
```

Etter kjøring ser fila *songlist.txt* slik ut:

```
Spinning wheel;Blood,Sweat and Tears;Rock;5:15
In my life;The Beatles;Pop;3:17
Rock me baby;B.B. King;Blues;3:28
Waiting for Margaux;Al Stewart;Pop;4:35
Folsom Prison blues;Johnny Cash;Country;2:42
Turning of the tide;Richard Thompson;Pop;2:58
All around my hat;Steeleye Span;Folk;4:09
Hell's bells;AC/DC;Rock;5:13
In my life;Joseph Vincent;Pop;1:46
Africa;Toto;Pop;4:56
```

Løsning:

```
def update_file(filename,song):
    f = open(filename,'a')
    f.write('\n'+ ';'.join(song))
    f.close()

def add_song_to_file(filename):
    song = enter_song()
    sangliste = read_file(filename)
    eksisterer = False
    for i in range(len(sangliste)):
        if song[0] == sangliste[i][0] and song[1] == sangliste[i][1]:
            print(f'{song[0]} already exists in {filename}')
            eksisterer = True
            break
    if not eksisterer:
        update_file(filename, song)
        print(f'{song[0]} has been added to {filename}')
```