Semesteroppgave 1

INF621 - Høstsemesteret 2021

Sammendrag

Semesteroppgavene er obligatoriske, og må være godkjente for å bestå INF621. Du skal levere én zip-fil, `sem1.zip`, som inneholder filene `oppg1.py`–`oppg3.py`. For å komprimere en eller flere filer til en zip-fil høyreklikker du filene (i dette tilfellet `oppg1.py`–`oppg3.py`) i maskinens filnavigasjonsprogram og velger `Komprimer` eller `Send til` → `Komprimert mappe`.
Frist: Torsdag 18. november kl 23:59

Du skal anta at alle tekstfilene i oppgavene ligger i samme mappe som pythonfilene. Etter at du har levert oppgaven anbefaler vi at du gjør noen eksperimenter med `pathlib` for å se hvordan det er å arbeide med filer som ligger i andre mapper. Filene som det refereres til i oppgavene finner du på `mitt.uib.no`.

1 Les fra/skriv til fil (30%)

Svar leveres på fil med navn `oppg1.py`. Denne oppgaven er en øvelse i å lese fra/skrive til en tekstfil.

1.a
(10 Poeng) Lag en funksjon `fil_info(filnavn)` som printer hvor mange tegn, ord og linjer filen med navn `filnavn` inneholder. Linjeskift telles ikke som tegn.

```
In [1]: fil_info('test.txt')
Det er 56 tegn, 12 ord og 7 linjer i filen test.txt.
```

Hint: Bruk metoden `split()` fra INF620 for å telle antall ord.

1.b
(10 Poeng) Lag en funksjon `tidspunkt()` som oppretter en fil med navnet `nå.txt`. Denne filen skal inneholde informasjon om tidspunktet filen ble opprettet på. Bruk formatet `'Fil opprettet ukedag (d)d. månedsnavn yyyy klokken TT:MM:SS.'`. Hvis `tidspunkt()` kjøres den 18.11.21 kl. 15:59:55 skal filen `nå.txt` inneholde:

```
Fil opprettet torsdag 18. november 2021 klokken 15:59:55.
```

Hint: Bruk `datetime` og `datetime.now()` fra forrige øving.

1.c
(10 Poeng) Lag en funksjon `antall(filnavn)` som returnerer en ordbok med bokstaver fra `'a'` til `'å'` som nøkler og antallet forekomster av de ulike bokstavene i `filnavn` som verdi.

```
In [2]: antall('test.txt')
Out[2]: {'a': 1, 'b': 0, 'c': 0, 'd': 4, 'e': 10, 'f': 1,
'g': 0, 'h': 2, 'i': 4, 'j': 0, 'k': 3, 'l': 2, 'm': 0, 'n': 5,
'o': 2, 'p': 0, 'q': 0, 'r': 2, 's': 2, 't': 7, 'u': 1, 'v': 1,
'w': 0, 'x': 0, 'y': 0, 'z': 0, 'æ': 0, 'ø': 0, 'å': 0}
```

Som et alternativ til å kjøre koden fra konsollen, kan du på slutten av filen `oppg1.py` legge til

```
if __name__ == '__main__':
    fil_info('test.txt')
    tidspunkt()
    forekomst = antall('test.txt')
    print('Bokstaver som forekommer i fila test.txt sortert etter avtakende forekomst:')
    for bokstav in sorted(forekomst, key=forekomst.get, reverse=True):
        f = forekomst[bokstav]
        if f>0:
            print('%s: %4d' % (bokstav, f))
```

2 Fotoboks (40%)

I denne oppgaven skal vi behandle data fra en simulert fotoboks. Fotoboksen lagrer data om biler som passerer en veistrekning med to felt. Dataen som fotoboksen har samlet opp finner du i filen `fotoboks.txt`. Du bør gjøre deg kjent med innholdet i filen før du begynner å løse oppgavene.

Hver linje i filen `fotoboks.txt` beskriver en bil som passerer fotoboksen. En passering er beskrevet med et tidspunkt, et skiltnummer og farten til bilen (målt til nærmeste hele km/t). Den samme bilen kan passere fotoboksen flere ganger. To passeringer kan være loggført til nøyaktig samme tidspunkt. Den samme bilen kan ikke passere fotoboksen to ganger til samme tidspunkt.

2.a
(10 Poeng) Vi vil lese informasjon fra filen `fotoboks.txt` og lagre den i en ordbok.

Forklar kort (ca. 1 setning per svar) hvorfor hverken passeringstidspunkt eller registreringsnummer bør brukes som nøkkel alene i denne ordboken. Forklar også hvorfor det er trygt å bruke tupler `(passeringstidpunkt, registreringsnummer)` som nøkler i ordboken. Lag en funksjon `svar_paa_2a()` som printer svaret ditt:

```
In [4]: svar_paa_2a()
Tidspunkt bør ikke brukes som nøkkel alene fordi (ditt svar)
Bilnummer bør ikke brukes som nøkkel alene fordi (ditt svar)
Tuplen kan brukes som nøkkel fordi (ditt svar)
```

2.b

(10 Poeng) Hver av linjene i filen `fotoboks.txt` er skrevet på det følgende formatet: `'dd.mm.yyyy-TT:MM:SS registreringsnummer fart'`. Ingen av registreringsnummerene inneholder mellomrom.

Lag en funksjon `hent_fotoboksdata()` som leser fra filen `fotoboks.txt` og som returnerer informasjonen lagret i denne filen i form av en ordbok. Nøklene i ordboken skal være tupler bestående av passeringstidspunkt (i form av et `datetime` objekt) og et registreringsnummer (i form av en streng). Verdien til nøklene skal være farten bilen hadde under denne målingen, målt til nærmeste hele kilometer pr. time.

```
In [5]: målinger = hent_fotoboksdata()
In [6]: målinger[(datetime(2021, 11, 11, 12, 24, 28), 'ST181525')]
Out[6]: 80
```

Hint: Bruk metoden `split()` til å dele opp hver enkelt linje.

2.c

(10 Poeng) La `målinger` være en ordbok lik den som skal bli returnert av funksjonen `hent_fotoboksdata()` og la `start_t` og `slutt_t` være to `datetime` objekter. Lag en funksjon `antall_bøter(start_t, slutt_t, målinger)` som returnerer antall passeringer i `målinger` som ble loggført i tidsrommet fra (og med) `start_t` til (men ikke med) `slutt_t` hvor farten til kjøretøyet var over 80 km/t.

```
In [7]: start_t = datetime(2021, 11, 8, 0, 0, 0)
In [8]: slutt_t = datetime(2021, 11, 15, 0, 0, 0)
In [9]: antall_bøter(start_t, slutt_t, målinger)
Out[9]: 267
```

Hint: Datetime objekter kan sammenlignes med $<$ og $<=$.

2.d

(10 Poeng) Lag en funksjon `fartsbøter(målinger)` som oppretter en fil som heter `bøter_per_dag.txt`. Denne filen skal inneholde informasjon om antall ganger et kjøretøy passerte fotoboksen med en hastighet over 80 km/t på de ulike dagene fra og med 8. november 2021 til og med 14. november 2021. Om du skriver følgende i konsollen:

```
In [10]: målinger = hent_fotoboksdata()
In [11]: fartsbøter(målinger)
```

skal filen `antall_bøter_per_dag.txt` se slik ut:

```
Mandag 08. november: 19 bøter
Tirsdag 09. november: 17 bøter
Onsdag 10. november: 39 bøter
Torsdag 11. november: 36 bøter
Fredag 12. november: 45 bøter
Lørdag 13. november: 60 bøter
Søndag 14. november: 51 bøter
```

Som et alternativ til å kjøre koden fra konsollen, kan du på slutten av filen `oppg2.py` legge til

```
if __name__ == '__main__':
    svar_paa_2a()
    målinger = hent_fotoboksdata()
    fartsbøter(målinger)
```

3 Kryptering (30%)

I denne oppgaven skal vi øve enda litt mer på å lese fra og å skrive til filer, denne gangen ved å se på kryptering. Å kryptere betyr å forsøke å gjøre en melding uleselig for alle andre enn de som er ment å motta den.

I denne oppgaven skal vi se på en type kryptering som kalles "monoalfabetisk substitusjon". Til tross for det lange navnet så er dette en ganske enkel (men "utrygg") måte å kryptere tekst på, hvor man erstatter bokstavene i en tekst med andre bokstaver. Vi antar at både den som krypterer og den som dekrypterer meldingen kjenner til hvordan bokstavene byttes ut med hverandre.

3.a
(10 poeng) Filen `hemmelig.txt` gir en oppskrift på hvordan du skal kryptere meldinger. Gjør deg derfor først kjent med innholdet i filen `hemmelig.txt`.

Linjene i `hemmelig.txt` er på formen `` `"bokstav1" erstattes med "bokstav2"' ``. I denne oppgaven skal du lage en funksjon `lag_krypteringsordbok()` som leser innholdet i filen `hemmelig.txt` og som returnerer en ordbok som vi senere skal bruke til krypteringen. Nøklene i denne ordboken skal være bokstaver og verdien skal være bokstaven som nøkkelen skal erstattes med i den krypterte meldingen.

```
In [10]: les_krypteringsordbok()
Out[10]: {'a': 'x', 'b': 'q', 'c': 'w', 'd': 'z', 'e': 'd',
'f': 'b', 'g': 'g', 'h': 'y', 'i': 'u', 'j': 'n', 'k': 'k',
'l': 'i', 'm': 't', 'n': 'e', 'o': 'r', 'p': 'f', 'q': 'l',
'r': 'v', 's': 'm', 't': 'o', 'u': 'c', 'v': 'j', 'w': 'p',
'x': 'h', 'y': 'a', 'z': 's'}
```

3.b

(10 poeng) Lag en funksjon `krypter(filnavn, krypteringsordbok)` som leser inn en fil med navn `filnavn` og oppretter en ny fil med navnet `'kryptert_' + filnavn`. Denne filen skal inneholde den krypterte teksten fra filen `filnavn`, hvor teksten har blitt kryptert ved hjelp av ordboken `krypteringsordbok` av samme type som den du leste inn i oppgave **3.a**. Dersom du kjører koden:

```
In [11]: krypteringsordbok = lag_krypteringsordbok()
In [12]: krypter('test.txt', krypteringsordbok)
```

skal det opprettes en fil med navn `kryptert_test.txt` hvor det står:

```
ydu
zdood dv de
iuode odkmobui yjrv
zc
kxe odmod krzde

zue
```

3.c

(10 poeng) Lag en funksjon `dekrypter(filnavn, krypteringsordbok)` som leser inn en fil med navn `filnavn` som inneholder tekst som har blitt kryptert med ordboken `krypteringsordbok`. Funksjonen `dekrypter(filnavn, krypteringsordbok)` skal opprette en ny fil med navn`'dekryptert_' + filnavn`. Dersom du kjører:

```
In [13]: krypteringsordbok = les_krypteringsordbok()
In [14]: dekrypter('kryptert_test.txt', krypteringsordbok)
```

skal det opprettes en fil med navn `dekryptert_kryptert_test.txt` hvor det står:

```
hei
dette er en
liten tekstfil hvor
du
kan teste koden

din
```

Som et alternativ til å kjøre koden fra konsollen, kan du på slutten av filen `oppg3.py` legge til

```
if __name__ == '__main__':
    krypteringsordbok = lag_krypteringsordbok()
    krypter('test.txt', krypteringsordbok)
    dekrypter('kryptert_test.txt', krypteringsordbok)
```

4 BONUS: Kodeknekker (0%)

I denne oppgaven må du arbeide mer selvstendig enn i de tidligere oppgavene. Om du har lyst til å utfordre deg selv litt ekstra kan du gjøre et forsøk på å knekke en kode. Du skal ikke levere noen egen fil for denne bonusoppgaven. Dersom du klarer å finne krypteringsordboken som ble brukt til å kryptere teksten i filen `kryptert_melding.txt` så kan du legge til en funksjon `bonus()` til slutt i python-filen `oppgave3.py` som returnerer denne ordboken.

I denne siste (frivillige) bonusoppgaven skal vi se hvor utrygt det er å kryptere meldinger med 'monoalfabetisk substitusjonchiffer". Dette skal vi vise ved å bryte krypteringen på innholdet i filen med navn `kryptert_melding.txt`. Metoden vi bruker for å bryte krypteringen heter frekvensanalyse. Du kan lese mer om denne teknikken på wikipedia: `https://no.wikipedia.org/wiki/Frekvensanalyse_(kryptografi)`

4.a
(0 poeng) Første steg er å finne frekvensen til hver av bokstavene i tekstfilen `kryptert_melding.txt`. Det er lurt å skrive ut bokstavene og frekvensene i sortert rekkefølge.

Hint: Her kan du bygge videre på funksjonen fra oppgave **1.c**.

4.b
(0 poeng) Den krypterte meldingen er en engelsk setning + en kjent engelsk bok. I en typisk engelsk tekst er bokstaven e mest vanlig og z minst vanlig. Alfabetet sortert etter sjeldenhet ser slik ut: e, t, a, o, i, n, s, h, r, d, l ,c, u, m, w, f, g, y, p, b, v, k, j, x, q, z. Om du er interessert i de eksakte frekvensene til hver bokstav kan du finne dem her: `https://en.wikipedia.org/wiki/Letter_frequency`.

Samenlign frekvensene i den krypterte teksten med frekvensen av de ulike bokstaver i en typisk engelsk tekst. Bruk denne informasjonen til å gjøre en kvalifisert gjettning på hvilken ordbok som er brukt til å kryptere meldingen.

4.c
(0 poeng) Test ordboken du gjettet på ved å bruke den til å "dekryptere" filen `kryptert_melding.txt`. Sannsynligvis gjettet du ikke helt rett, men du klarer kanskje å gjøre små endringer på ordboken som gir et litt bedre svar. De 70-100 første tegnene i `kryptert_melding.txt` vil være spesielt nyttige for deg. Fortsett å endre på ordboken til du har klart å knekke koden.

Hint: Ta vare på tidligere gjetninger slik at du ikke må begynne på nytt dersom du gjør en feil. Let etter ord i den "nesten" dekrypterte teksten som ligner på vanlige engelske ord som "it", "the" og 'was" for å finne ut hvor du har gjettet feil.