Programmering og Modellering (PoM)
Ugeseddel 4 — Uge 39 — Deadline Ingen aflevering

Kim Steenstrup Pedersen, Katrine Hommelhoff Jensen, Knud Henriksen,
Mossa Merhi og Hans Jacob T. Stephensen

18. september 2014

4 Plan for ugen

I programmer har vi hidtil benyttet *simple variable*, der hver kun kan rumme en enkelt værdi, men nu skal vi arbejde med *sekvenser*, der med et enkelt navn kan dække en hel stribe data.

I denne uge fokuseres på Pythons forhåndsdefinerede datastrukturer som inkluderer *tegnstrenge* og *lister*, *hash-tabeller* (eng. *dictionaries*), *sæt* (eng. *tuples*). Man tilgår elementerne i en sekvens ved *indicering* eller ved at udtage en *skive* (eng.: *slice*). Tegnstrenge er *uforanderlige* (eng.: *immutable*), hvilket vil sige, at man ikke kan ændre dele af en tegnstreng, mens man i lister både kan hente (læse) og ændre (skrive) enkelte elementer eller hele skiver.

Vi skal ligeledes se hvorledes vi kan indlæse fra og skrive til *filer* (eng. *files*) i Python. Endelig skal vi se hvordan vi kan lave grafiske illustrationer i Python — en central kundskab indenfor naturvidenskab.

I denne uges tirsdagsforelæsning gennemgås de forskellige datastrukturer og deres anvendelser. Herunder repræsentation af tegn i Unicode, og hash-tabeller, som muliggør effektive tabelopslag.

I torsdagsforelæsningen behandles sæt (eng. tuples) og filer. Desuden omtales funktionen `matplotlib.pyplot.plot()` til grafisk fremstilling i et koordinatsystem.

Forberedelse til forelæsningerne:

Til tirsdag:

Læs Guttag kap. 4.6, 5 (eks. 5.3) samt noterne om "Tegn og tegnstrenge".

Til torsdag:

Læs Guttag kap. 4.6, 5 (eks. 5.3). Læs også notatet om Visualisering med `pyplot`.

Bemærk: Der er ingen obligatorisk afleveringsopgave i denne uge.

Biblioteksynonym Programbiblioteket `matplotlib.pyplot` betegnes i det følgende blot `plt`, sådan som det for eksempel vil gælde efter python-sætningen

```
import matplotlib.pyplot as plt
```

4.1 Tirsdagsøvelser

Besvarelser af disse opgaver skal ikke afleveres, men opgaverne forventes løst inden torsdag.

4ti1 Prøv ved hjælp af oplysningerne i noterne om tegn og tegnstrenge at gøre rede for de besynderlige koder i nedenstående eksempel:

```
>>> unitekst = u'\u201Cog s\xe5 videre \u2026\u201D'
>>> unitekst
u'\u201cog s\xe5 videre \u2026\u201d'
>>> print unitekst
```

```
“og så videre ...”
>>> tekst = unitekst.encode('utf8')
>>> tekst
'\xe2\x80\x9cog s\xc3\xa5 videre \xe2\x80\xa6\xe2\x80\x9d'
>>> unitekst == unicode(tekst,'utf8')
True
```

4ti2 En samling mønter kan bekvemt repræsenteres af en associationsliste, der har møntværdier som indgangsnøgle og antallet af mønter af den pågældende slags som indhold. To tiere, en tokrone, en enkrone og tre halvtredsører repræsenteres for eksempel af {`1000: 2, 200: 1, 100: 1, 50: 3`}.

Skriv en funktion, som beregner det samlede beløb i en sådan repræsentation. For eksemplet ovenfor skal funktionsværdien være 2450 (øre).

Skriv også en funktion, der omvendt for et argument i form af et ørebeløb som funktionsværdi giver en associationsliste, der repræsenterer dette beløb under brug af færrest mulige mønter. For 2450 kunne funktionsværdien for eksempel være {`2000: 1, 200: 2, 50: 1`} eller {`2000: 1, 1000: 0, 500: 0, 200: 2, 100: 0, 50: 1`}.

[Vink: Denne sidste funktion kan konstrueres efter det såkaldt *grådige princip*, hvor de forskellige mulige danske møntværdier afprøves efter tur med den største først.]

4ti3 Ved *rækkevis opstilling* af en sekvens `q u i c k b r o w n f ...` forstås den systematik, hvor rækker udfyldes en ad gangen ovenfra, og man først, når en række er fuld, fortsætter i den næste:

q	u	i	c
k	b	r	o
w	n	f	...
...			

Ved *søjlevis opstilling* udfyldes søjlerne fra venstre, og først når en søjle er fuld, fortsætter man i den næste:

q	c	r	n	⋮
u	k	o	f	
i	b	w	⋮	

Inden for et rektangulært skema med r rækker og s søjler er et element ved rækkevis opstilling af en sekvens havnet i række i og søjle j. Hvor ville det samme element være havnet ved søjlevis opstilling af sekvensen?

[Vink: Nummerer såvel sekvensen som rækker og søjler begyndende med 0. Overvej, om heltalsdivision (*floor division*) kunne være til nytte.]

4.2 Torsdagsøvelser

Besvarelser af disse opgaver skal ikke afleveres, men opgaverne forventes løst inden tirsdag i efterfølgende uge.

4to1 Definer funktionen `has_duplicates` som tager en liste som argument og returnere `True`, hvis listen indeholder mindst et element som forekommer flere gange i listen.

Hvad skal `has_duplicates([])` returnere?

4to2 Brug hashtabeller til konstruktion af en mere effektiv udgave af funktionen `has_duplicates` fra opgave 4to1.

4to3 Skriv en funktion som tager en liste af tal som argument og returnerer den kumulative sum, dvs. en ny liste hvor det i'te element er summen af de $i+1$ foregående elementer i den originale liste. Eksempel: Den kumulative sum af `[1, 2, 3]` er `[1, 3, 6]`.

For en liste $[a_0, a_1, a_2, \ldots, a_{n-1}]$ af længde n udtrykkes resultatet $[a_0, a_0 + a_1, a_0 + a_1 + a_2, \ldots, a_0 + a_1 + a_2 + \ldots + a_{n-1}]$ med $\frac{n(n-1)}{2}$ additionstegn, men faktisk er det muligt at

beregne resultatlisten med kun $n-1$ additionsoperationer! Hvor mange additionsoperationer bruger din løsning?

4to4 Definer under brug af formatoperatoren `%` en funktion `visBetaling(`h`)`, sådan at hvis `htabel` er en hashtabel af den slags, der blev benyttet i opgave 4ti2, og som altså knytter antal til øreværdier, så vil kaldet `visBetaling(htabel)` udskrive en pæn opstilling med antal og møntværdi i separate linjer for hver mønt i hashtabellen.

Funktionskaldet

```
visBetaling({2000: 1, 1000: 0, 500: 0, 200: 2, 100: 0, 50: 1})
```

kunne for eksempel give anledning til udskrivning af

```
Betal med  0 mønter à  100 øre
Betal med  0 mønter à 1000 øre
Betal med  1 mønter à 2000 øre
Betal med  2 mønter à  200 øre
Betal med  1 mønter à   50 øre
Betal med  0 mønter à  500 øre
```

(Som eksemplet viser, er det helt i orden at lade hashfunktionen bestemme linjernes rækkefølge og at blæse på dansk grammatik i denne opgave.)

4to5 Definer en funktion `parallelleLister(`d`)` af en formel parameter, som omsætter en hashtabel til et par af lister. Hvis hashtabellen knytter x'er til y'er, skal funktionen returnere parret bestående af listen af x'er og listen af de tilsvarende y'er, idet listen af x'er skal være i ikke-aftagende orden.

`parallelleLister({'a': 0, 'c': 5, 'd': 0, 'b': 8})` skal for eksempel returnere `(['a', 'b', 'c', 'd'], [0, 8, 5, 0])`.

4to6 Vælg et passende sted på din computer til eksperimenter med filer. Lad os sige, du vælger kataloget med absolut stinavn ***`katalog`***.

Indtast i et Python-afviklingsvindue følgende linjer:

```
sti = 'katalog'
udfil = open(sti + '/udgangsfil.txt', 'w')
udfil.write('Dette er en prøve!\n')
udfil.flush()
```

og åbn derefter den dannede fil i et passende tekstbehandlingsprogram (for eksempel Notesblok, TextEdit, gedit eller emacs) for at kontrollere, at teksten faktisk er blevet skrevet i den.

Afhængigt af, hvordan tegntabellerne er sat op på maskinen, kan det være nødvendigt med særbehandling af de danske bogstaver, for eksempel med

```
udfil.write('Dette er en pr\xf8ve!\n')
```

Prøv at skrive noget mere ud til filen, og bemærk, hvordan tekst først kommer til syne i tekstbehandlingssystemets vindue efter `udfil.flush()`. Måske skal tekstvinduet genindlæses (eller lukkes og genåbnes). Husk til sidst at skrive

```
udfil.close()
```

4to7 Denne opgave bruger det samme katalog ***`katalog`*** som den foregående opgave. Opret i dette katalog med et passende tekstbehandlingsprogram en fil med navnet `indgangsfil.txt`. Tast 3–5 linjers tekst ind, og gem filen (som rå eller "flad" tekst, eng. *plaintext*).

Knyt nu Python-navnet `indfil` til filen med

```
sti = 'katalog'
indfil = open(sti + '/indgangsfil.txt')
linje = indfil.readline()
print linje
```

Fortsæt med kald af `indfil.readline()`, til alle linjer i filen er læst. Husk til sidst at skrive

```
indfil.close()
```

4to8 Definer en funktion `danHashtabel(`*filnavn*`)` af en formel parameter, sådan at hvis et kald af formen `danHashtabel(`*filnavn*`)` udføres på en computer, hvor *filnavn* er absolut stinavn til en tekstfil med indhold som beskrevet nedenfor, så returneres en hashtabel med tekstfilens indhold.

Tekstfilen antages at bestå af n linjer, hver med to flydende talord adskilt af komma

x_1`,` y_1
...
x_n`,` y_n

og talparrene skal returneres som en hashtabel $\{x_1$`:` y_1`, ...,` x_n`:` $y_n\}$.

[Vink: Man kan bruge `eval()` for at få omsat en tekstlinje af formen `'`x`,` y`\n'` til et Python-talpar.]

Afprøv funktionen på filerne `flueaeg.txt` og `pattedyr.txt`, som ligger på kursushjemmesiden i undermappen Datamateriale under Undervisningsmateriale. (Overfør først de to filer til din egen computer.)

4to9 Kombiner funktionerne

- `plt.plot()`
- `parallelleLister()` fra opgave 4ti5
- `danHashtabel()` fra opgave 4to8
- eventuelt `plt.show()`

så observationerne fra `flueaeg.txt` og `pattedyr.txt` kan vises grafisk. Læg hver observation (x, y) ind i koordinatsystemet som en lille farvet cirkelskive[1].

[Vink: Husk, at hvis `xryr` er et par af lister, der skal overføres som de to første parametre til `plt.plot()`, må man skrive `plt.plot(* xryr,...)`.]

[1] En sådan afbildning kaldes for et *scatter plot* eller en *scattergraph*. Andre betegnelser er *scatter chart*, *scatter diagram* eller *scattergram*. Ordet *scatter* er engelsk for at sprede eller strø, og sammenfaldet med betegnelsen for pythons `scatter`-operator er tilfældigt.