Høgskolen i Østfold

Python

Et utdrag fra: http://www.ia.hiof.no/~borres/ml

Dette utdraget fra et relativt dynamisk vevsted vil alltid være litt ute av synk. Det er bare ment som litt trøst for de som vil ha noe å lese på senga før de sovner.

Sidene er produsert ved hjelp av WXT (http://www.ia.hiof.no/~borres/wxtdoc/), Python og PrinceXML (http://www.princexml.com/)

Rettighetene er de samme som for vevstedet materialet er hentet fra:

- Bruk det til hva du vil så lenge du refererer til vevstedet
- Det følger ikke med noen som helst garanti for at koden og teksten er riktig i en hver forstand.



Innhold

- Python
- Py-Eksempler
- CGI
- DOM i Python
- SAX i Python
- Objekter
- Databaser
- ting og tang
- Pythontesting
- Bildegalleri
- Slideshow
- Temperaturer
- Data om land
- Hele verden
- Snart

Python

Hva:

Om Python

Alle programmeringsspråk har sine sterke og svake sider. Hvilket språk vi velger bør være avhengig av hvilke oppgaver vi skal løse. Python er et "ubyråkratisk" språk som er godt egnet til tekstbehandling, lasting av nettressurser, enkle beregninger og til CGI-programmering. Python er et ganske komplett objektorientert språk og kan brukes, alene eller sammen med biblioteker skrevet i andre språk, til det meste. Bruken i dette materialet utnytter ikke alle disse mulighetene. På den annen side er det ikke noe vi gjør i dette materialet som vi ikke kan gjøre i andre språk som C, C++, Java, C#.

Python er valgt fordi det gjøre en del "typiske" ting enkelt og fordi språket er stabilt og kan kjøres på alle aktuelle plattformer. Vi kan f.eks. grovteste CGI-programmene våre på en Windows-klient og så flytte dem over på en Linux-server.

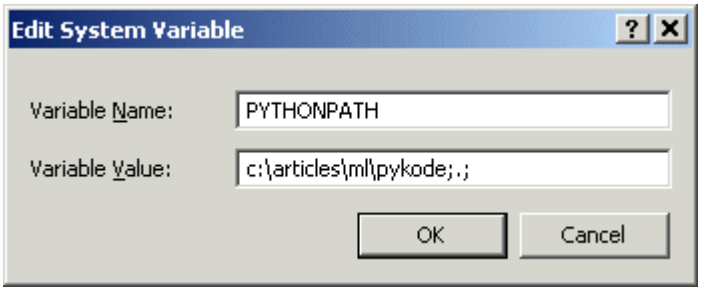
Dette er ikke noen lærebok i Python. Det er tilgjengelig en rekke bøker (se referenser nedenfor) og nettressurser for å lære Python.

Jeg arbeider med den siste versjonen jeg får tak i, på Windows XP og jeg bruker Pythons egen IDLE-editor. Det er altså de omgivelsene som dukker opp når jeg starter Python under Windows. Det er mange andre måter å organisere arbeid med Python på enn det jeg har som referanse her. CGI-skriptene jeg lager kjøres på en Linux-tjener.

PYTHONPATH

Vi må fortelle operativsystemet hvor vi akter å plassere våre Pythonmoduler. På samme måte som Java har sin classpath, har Python sin pythonpath. Måten å sette dem opp på er helt parallelle. Jeg åpner Settings/Controlpanel og starter System. Her velger jeg Advanced og klikker på knappen: Environment Variables. Så leter jeg etter PYTHONPATH i lista med overskriften System variables. Hvis den finnes så ber jeg om å få editere (EDIT) den. Hvis den ikke finnes så lager jeg ny (NEW).

På min maskin ser det i øyeblikket slik ut:



Jeg har alle mine Pythonmoduler samlet i en katalog: **c:\articles\ml\pykode**

Syntaksen i pathlista er slik at alle pather er adskilt med ; Husk at . representerer pather som er satt andre steder. Ta alltid med . - pathen, dersom du er i tvil.

Å editere en modul

Jeg pleier å la Python administrere editorvinduene mine. Det vil si at jeg går i menyen File og velger Open Module. Hvis PYTHONPATH er satt riktig, vil IDLE finne fram den riktige fila når jeg skriver inn navnet på en eksisterende modul, f.eks. bokxml. Jeg får da et nytt Pythonvindu med mange av de samme egenskapene som det første vinduet som åpner seg som IDLE. Det er selvsagt ikke noe i veien for å editere i en annen editor.

I alle fall kjører jeg testing fra IDLE, som vist nedenfor.

Testing av modulen

Starten på en arbeidssekvens der jeg arbeider med modulen boktoxml kan se slik ut, i IDLE-vinduet:

```
Python 2.2 (#28, Dec 21 2001, 12:21:22) [MSC 32 bit (Intel)] on win32
Type "copyright", "credits" or "license" for more information.
IDLE 0.8 -- press F1 for help
>>> import boktoxml
>>> def test():
        reload(boktoxml)
        bokxml.produce("c:\\articles\\ml\\python\\bokliste.txt")

>>> test()
>>> test()
>>> test()
```

Jeg lager meg en testprosedyre, test, for anledningen. Den automatiserer to ting: oppdatering av siste versjon av modulen jeg tester og et kall på den aktuelle funksjonen med den eller de parametrene jeg bruker som testgrunnlag. Den typiske arbeidssyklusen blir da å:

- redigere i det vinduet jeg har kildekoden, boktoxml.py
- kalle test() i IDLE-vinduet
- studere resultatet hvor det måtte dukke opp, f.eks i en nettleser eller som feilmelding i IDLE-vinduet
- redigere i det vinduet jeg har kildekoden, boktoxml.py
- osv

Referanser og eksempler:

Pythonbøker

- Beginning Python, From novice to Professional, Magnus Lie Hetland
  Apress, ISBN 1-59059-519-X
- Python, The Complete Reference,Martin C Brown
  Mc Graw Hill, ISBN 0-07-212718-X
- XML Processing with Python, Sean McGrath
  Prentice Hall, ISBN 0-13-021119-2
- Learning to Program Using Python, Alan Gauld
  Addison Wesley ,ISBN 0-201-70938-4
- Dive Into Python, Mark Pilgrim
  Apress, 2004,ISBN:1590593561
  også på nettet: diveintopython.org/
- Foundations of Python Network Programming, John Goerzen
  Apress, 2006,ISBN:1590593715

Pythondokumentasjon
Python 2.3 Quick reference: rgruet.free.fr/PQR2.3.html

Pythoneksempler

transformasjon av filformater

Hva:

Demonstrasjon av noen momenter ved utvikling i Python

HTML fra en enkel tekstfil

Programmet leser inne en enkel tekstfil og plasserer innholdet i en pre-tag på en minimalistisk HTML-side. HTML-siden ligger som en tekst i programmet.

Input fil: http://www.ia.hiof.no/~borres/ml/python/frej1_txt.html
Output fil: http://www.ia.hiof.no/~borres/ml/python/frej1.html
Python koden:

```
"""
This is a modul that reads a textfil
wraps it in a HTML-skeleton and produce
a nice HTM-page
"""

#------------------------
# HTML-fragments
HTML_PAGE="""<html>
<head>
    <title>a page</title>
</head>
<body>
<h1> Farbror Frej:</h1>
<pre>
%s
</pre>
</body>
</html>
"""

#------------------------
# filenames
infile='c:\\tull\\frej1.txt'
outfile='c:\\tull\\frej1.html'


#------------------------
# Read / write text files

def getTextFile(filename):
    try:
        file=open(filename,'r')
        res=file.read()
        file.close()
        return res
    except:
        print 'Trouble reading: '+filename
        return None

def storeTextFile(filename,txt):
    try:
        file=open(filename,'w')
        file.write(txt)
        file.close()
    except:
        print 'Trouble writing to: '+filename

#------------------------
# do the job

def doit():
    txt=getTextFile(infile)
    if txt!=None:
        txt=HTML_PAGE%txt
        #print txt
        storeTextFile(outfile,txt)

#doit()
```

Transformasjon: CSV-XML

Modulen boktoxml gjør noen av de grunnleggende operasjonene som inngår i å lage en XML-fil fra en kommaseparert fil:

- åpner og leser en fil på en sivilisert måte
- splitter innholdet opp i linjer, og forkaster meningsløse linjer
- splitter hver linje i kommeseparete deler
- bruker delene til å produsere XML-elementer
- skriver alt tilbake til fil på en sivilisert måte

Input fil: http://www.ia.hiof.no/~borres/ml/python/bokliste_txt.html
Output fil:http://www.ia.hiof.no/~borres/ml/python/bokliste.xml
Pythonkoden:

```
# transform a commaseparated (CSV) file to XML
# Raw data as lines:
# title,author,publisher,year,isbn,pages,course,category,comment

#----------------------------
# XML-skeletons
# a template for a xml-fragment
XMLFragment="""
<book isbn="%s" pages="%s">
      <title>%s</title>
      <course>%s</course>
      <category>%s</category>
      <author>%s</author>
      <publisher>%s</publisher>
      <year>%s</year>
      <comment>%s</comment>
</book>
"""

# a template for a complete xml-file
XMLFile="""<?xml version="1.0" encoding="ISO-8859-1"?>
<booklist>
%s
</booklist>
"""
#-----------------------------------
# read / write text files
def getTextFile(filename):
    try:
        file=open(filename,'r')
        intext=file.read()
        file.close()
        return intext
    except:
        print 'Error reading file ',filename
        return ''

def storeTextFile(filename,txt):
    try:
        outfile=open(filename,'w')
        outfile.write(txt)
        outfile.close()
    except:
        print 'Error writing file ',filename

#--------------------------------
# produce and save XML
def makeXML(filename='c:\\articles\\ml\\python\\bokliste.txt'):
    # les en text fil
    text=getTextFile(filename)
    if(text==''):
        return
    content=''
    # plukk ut linjene
    lines=text.split('\n')
    for line in lines:
        line.strip()
        # drop tomme linjer og kommentarlinjer
        if(len(line)<2):
            continue
        if(line[0:2]=='//'):
            continue
        # har en boklinje, finn delene
        pcs=line.split(',')
```

```
        if(len(pcs)!=9):
            print 'ignore:' , line
            continue
        content+=XMLFragment%(pcs[4],pcs[5],pcs[0],pcs[6],
                              pcs[7],pcs[1],pcs[2],pcs[3],pcs[8])

    storeTextFile(filename.replace('.txt','.xml'),XMLFile%content)
```

Transformasjon: CSV-HTML

Modulen boktohtml tar den samme tekst-fila som i eksempelet ovenfor og transformerer den til en htmlfil som viser en liste av bøker med forfatter

Input fil: http://www.ia.hiof.no/~borres/ml/python/bokliste_txt.html
Output fil: http://www.ia.hiof.no/~borres/ml/python/bokliste.html
Python koden:

```
# transform a commaseparated (CSV) file to HTML
# Raw data as lines:
# title,author,publisher,year,isbn,pages,course,category,comment

#----------------------------
# XML-skeletons
# a template for a complete html-file
HTMLFile=
"""<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
    <meta HTTP-EQUIV="Content-Type"
          content="text/html;
          charset=iso-8859-1">
    <title>books</title>
    <style>
     li{margin-top:10;}
     .fat{font-weight:bold; color:red}
    </style>
</head>
<body>
<h1>Bokliste</h1>
<ul>
%s
</ul>
</body>
</html>
"""

# a template for a xml-fragment
HTMLFragment="""
    <li>
        <div class="fat">%s</div>
        <div>%s</div>
    </li>
"""

#-------------------------------------
# read / write text files

def getTextFile(filename):
    try:
        file=open(filename,'r')
        intext=file.read()
        file.close()
        return intext
    except:
        print 'Error reading file ',filename
        return ''

def storeTextFile(filename,txt):
    try:
        outfile=open(filename,'w')
        outfile.write(txt)
        outfile.close()
    except:
        print 'Error writing file ',filename


#--------------------------------
# produce and save HTML
def makeHTML(filename='c:\\articles\\ml\\python\\bokliste.txt'):
    # les en text fil
    text=getTextFile(filename)
```

```
if(text==''):
    return
content=''
# plukk ut linjene
lines=text.split('\n')
for line in lines:
    line.strip()
    # dropper tomme linjer
    if(len(line)<2):
        continue
    # dropper kommentarlinjer
    if(line[0:2]=='//'):
        continue
    # har en boklinje, finn delene
    pcs=line.split(',')
    if(len(pcs)!=9):
        print 'ignore:' , line
        continue
    content+=HTMLFragment%(pcs[0].strip(),pcs[1].strip())

storeTextFile(filename.replace('.txt','.html'),
              HTMLFile%content)
```

Transformasjon: XML-HTML

Modulen bokxmltohml tar xml-fila og transformerer den til html.

Input fil: http://www.ia.hiof.no/~borres/ml/python/bokliste.xml
Output fil: http://www.ia.hiof.no/~borres/ml/python/boklistefromxml.html
Pythonkode:

```
import xml.dom.minidom
#----------------------
# Produce a HTML-file from a XML-file
# http://www.ia.hiof.no/~borres/ml/python/p-python.html
# Input:
#<booklist>
#<book isbn="txt" pages="txt">
#<title>txt</title>
#<course>txt</course>
#<category>txt</category>
#<author>txt</author>
#<publisher>txt</publisher>
#<year>txt</year>
#<comment>txt</comment>
#</book>
#...
#</booklist>
#
# produce a html-file with same name, different ext
# Use: produceHTML(filename)
#-----------------------------------------------------------------

#--------------------------------------
# skeleton for HTML-file
HTMLFile=
"""<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
    <html>
    <head>
    <META http-equiv="Content-Type" content="text/html;
          charset=iso-8859-1\">
    <title>bokliste</title>
    <!-- produced by python -->
    </head>
    <body>
    <h2>Litteraturliste</h2>
    %s
    </body>
    </html>
    """

#--------------------------------------
# read / write text files
def getTextFile(filename):
    try:
        file=open(filename,'r')
        intext=file.read()
        file.close()
        return intext
    except:
        print 'Error reading file ',filename
        return ''

def storeTextFile(filename,txt):
    try:
        outfile=open(filename,'w')
        outfile.write(txt)
        outfile.close()
    except:
        print 'Error writing file ',filename

#----------------------------------
# collect all text in a node
def getText(nodelist):
    rc = ''
```

```
    for node in nodelist:
        if node.nodeType == node.TEXT_NODE:
            t=node.data.encode('ISO-8859-1')
            rc += t
    return rc

def getStrippedText(nodelist):
    rc = ''
    for node in nodelist:
        if node.nodeType == node.TEXT_NODE:
            t=node.data
            t=t.strip()
            t=node.data.encode('ISO-8859-1')
            if t!='\n':
                rc += t.strip()
    return rc

#---------------------------------------
# Produce a fragment for one book
def makeBook(q):

    result=q.getElementsByTagName('title')[0]
    T='<h3>%s</h3>\n'%getText(result.childNodes)

    T+='<div>isbn:%s</div>\n'%q.getAttribute('isbn').encode('ISO-8859-1')

    result=q.getElementsByTagName('course')[0]
    T+='<div>%s</div>\n'%getText(result.childNodes)

    result=q.getElementsByTagName('category')[0]
    T+='<div>%s</div>\n'%getText(result.childNodes)

    result=q.getElementsByTagName('author')[0]
    T+='<div>%s</div>\n'%getText(result.childNodes)

    result=q.getElementsByTagName('publisher')[0]
    T+='<div>%s</div>\n'%getText(result.childNodes)

    result=q.getElementsByTagName('year')[0]
    T+='<div>%s</div>\n'%getText(result.childNodes)

    result=q.getElementsByTagName('comment')[0]
    T+='<div>%s</div>\n'%getText(result.childNodes)

    return T

#----------------------------------
# Produce the entire file and save it
def produceHTML(infile='c:\\articles\\ml\\python\\bokliste.xml'):
    # load xml-file
    document=getTextFile(infile)
    if document=='':
        return

    # and establish DOM-tree
    dom = xml.dom.minidom.parseString(document)

    # get a list of all books
    bliste=dom.getElementsByTagName('book')

    # doing the books
    T=''
    for b in bliste:
        T+=makeBook(b)
```

```
    # write the file
    storeTextFile(infile.replace('.xml','fromxml.html'),HTMLFile%T)

    # clean up
    dom.unlink()
```

Du kan sammenligne denne Pythonkoden med en XSLT-fil som gjør samme jobben:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="xml"
     omit-xml-declaration="no"
     indent="yes"
     doctype-public="-//W3C//DTD XHTML 1.0 Strict//EN"
     doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"
     encoding="ISO-8859-1"/>

<xsl:template match="/">
    <html>
    <head>
    <title>bokliste</title>
    </head>

    <body>
    <h1>Litteratur</h1>
    <xsl:apply-templates select="booklist/book"/>
    </body>
    </html>
</xsl:template>

<xsl:template match="book">
   <h3><xsl:value-of select="title"/></h3>
   <div><xsl:value-of select="author"/></div>
   <div><xsl:value-of select="publisher"/>,
   <xsl:value-of select="year"/></div>
   <div>Isbn: <xsl:value-of select="@isbn"/></div>
   <div class="kommentar"><xsl:value-of select="comment"/></div>
</xsl:template>

</xsl:stylesheet>
```

Referanser og eksempler:

Alle aktuelle filer fra eksemplene er sitert i teksten

Python og CGI

Common Gateway Interface

Hva:

Demonstrasjon av noen momenter ved utvikling av CGI-script

Utgangspunktet er en situasjon der jeg som Windowsbruker som skal lage CGI-script på en Linux server. Det er et par ting som er verdt å merke seg. Det gjelder formen på selve scriptet, debugging og manipulasjon av rettigheter til de filene vi lage på tjeneren.

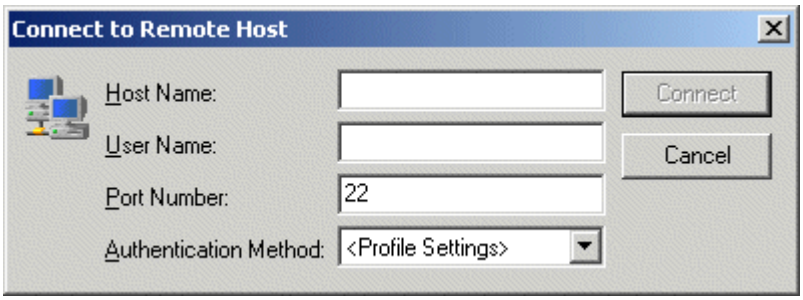
Jeg forutsetter at jeg kan nå filene på tjeneren på to måter.

1. Jeg kan se og manipulere kataloger og filer via File Explorer. Det vil si at Linux serveren er mappet opp mot min maskins filsystem via Samba, og at mitt område på serveren er blitt tildelt en bokstav som identifikator.
2. Jeg kan åpne et "Linux-vindu" mot serveren ved hjelp av SecureShell (eller et annet program).



Oppkopling

Når jeg starter SecureShell får jeg opp et vindu og jeg velger Quick Connect. Dialogboksen som skal fylles inn for å få kontakt ser slik ut:



Host Name er den serveren du vil kjøre mot, f.eks. **brage.hiof.no**
User Name er det brukernavnet du har på serveren, f.eks. **borres**
Som Authentication Method velger du Password. Når dette er klart trykker du Connect og svarer med riktig passord når du blir spurt.

Med mitt utgangspunkt trenger jeg å være koplet opp mot serveren via Secure Shell av to årsaker. For det første vil jeg kontrollere og sette riktige rettighter til de filene jeg skal arbeide med. For det andre kan jeg bruke det vinduet jeg har åpnet mot serveren til debugging.

Rettigheter

Sett rettighetene til 755

```
chmod 755 minfil.py
```

Det vil si at vi gir rettighetene: rwx r-x r-x. Alle kan lese og kjøre programmer, mens bare du kan skrive.

Dersom vi gir færre rettigheter får vi problemer med å få til den ønskede funksjonaliteten, og dersom vi gir flere rettigheter er det mulig at serveren protesterer. Grunnen til dette er at det kan være lagt inn sikkerhetskontroll så vi ikke åpner for at hvem som helst kan endre våre filer. Scriptet vårt nekter å kjøre, uten at vi får noen instruktiv feilmelding.

De som bruker UltraEdit under Windows kan støte på et lite problem. Dersom du har satt opp UE til å lage back-up filer, kan du oppleve at rettighetene ødelegges når du sparer fila. Du kan unngå dette ved enten å skru av back-up eller ved å legge til en linje i fila: winnt\uedit32.ini:

```
DontUseRename=1
```

Debugging

Siden scriptet vårt kjører på serveren, har vi ikke slik kontroll over hvor feilmeldinger havner som når vi kjører på vår egen maskin. Jeg pleier derfor å teste ut så mye som mulig av scriptet på min maskin før jeg laster det over på serveren.

Det er to måter å debugge serverskript i Python:

- Lese feilrapport fra serveren. Når serveren har problemer med å utføre en HTTP-forespørsel (som åkjøre et skript) lages det en feilrapport. Denne feilrapporten kan inneholde tekst som er generert fra serveren selv eller fra skripttolkningen i Python. Vi kan åpne et vindu i SSH og be om å få se denne rapporteringen fortløpende. Kommandoen som gir oss denne rapporteringen avhenger av hvordan serveren er satt opp. systemansvarlige skal kunne svare på det. Eksempler som angir hvordan en slik kommando under Linux kan være:
  ```
  tail -f /var/log/httpd/error_log
  tail -f /var/log/apache/error.log
  tail -f  /var/log/apache2/error.log
  tail -f /var/log/httpd/error_log | grep ml150
  ```
  Den siste avgenser rapporteringen til rapporter som gjelder bruker ml150
- Skru på "CGI Trace Back" i Python. Skriv følgende linje i starten av skriptet:
  ```
  import cgitb; cgitb.enable()
  ```
  Merk at det er ikke alle feil som traces. Alle feil som skyldes syntaksfeil som oppdages før skriptet startes rapporteres som server-feil, dvs. server har ikke klart å starte skriptet. Skru av Trace Back, fjern linja som oppgitt over, nå skriptet er ferdig testet og skal i produksjon.

CR-LF

Tekstfiler på Unix/Linux lagres med bare linjeskift(LF), mens Windows normalt lagrer filene med "vognretur" og linjeskift (CRLF). Normalt, f.eks. i nettlesere, skaper ikke dette noe problem siden de fleste programmer justerer for denne forskjellen. Shellet på Linux som skal tolke vår fil før den starter Python liker ikke CRLF i det hele tatt. Det vil si at vi må passe på at de filene vi legger ut som CGI-script på en Linux-server følger Linux standarden og bare har LF. Hvis vi editerer med Linuxverktøy på Linux er ikke dette noe problem, men hvis vi editerer på Windows med

Windowsverktøy må vi ta våre forholdsregler. Vi må ha en editor som legger fra seg filene slik Linux liker dem samtidig som de må vise dem fram på en fornuftig måte mens vi editerer.

Det er mange måter å ordne dette på. Jeg bruker Ultraedit. Ultraedit kan gjøre to nyttige ting i denne sammenhengen.

- transformere filer mellom DOS og Unix. Se menyen File/Conversions
- kan settes opp slik at den alltid skriver filer tilbake i den formen de ble åpnet. Se menyen Advanced/Configuration og tab General

Eksempel 1

Et minimumseksempel på et CGI-script kan være slik:

```
#! /usr/bin/python2

print 'Content-type: text/plain\n'
print 'hello'
```

Den første linja angir at denne file skal leses og kjøres av python2 som er plassert slik stien angir. Dette er en konvensjon i Unix/Linux. Filer som ligger på cgi-bin er kandidater for å eksekveres, og linux må finne ut hvem som skal gjøre det. Fileextensions spiller ikke samme rolle i Linux som i Windows. Den andre linja er obligatorisk i alle cgi-script og angir hva som kommer for det programmet som mottar resultatet. Dersom vi skal skrive HTML-kode bør vi skrive:

```
   print 'Content-type: text/html\n'
```

Nettlesere har variabel følsomhet for dette. Noen tolker html selv om vi angir text/plain.

Den tredje linja er din melding til verden. Vi ser at det er ikke noen koplett HML-fil, men siden nettleserne er så tilgivende vil de fleste presentere resultatet.

Gi denne fila et navn, f.eks.: mincgi.py og legg den ut på ditt serverområde under katalog cgi-bin. Lag denne katalogen hvis du ikke har den. Sett rettigheter som vist ovenfor. Test den ved å skrive URL'en i adressefeltet i en nettleser. Adressen kan være slik for bruker ml150:

```
http://www.ia-stud.hiof.no/~ml150/cgi-bin/mincgi.py
```

eller den kan være slik:

```
http://frigg.hiof.no/ml150/cgi-bin/mincgi.py.
```

Sjekk med systemansvarlig i de omgivelsene du jobber i.

Eksempel 2

I dette eksempelet kopler vi et script opp mot en HTML-side der vi skal fylle inn noen data og sende dem til scriptet. Vi lager en minimalistisk HTML-fil for enkelhetens skyld, merk at adressen til pythonscriptet er satt opp for bruker ml150 på serveren ml:

```
<html>
<head>
</head>
<body>
<form action="http://www.ia-stud.hiof.no/~ml150/cgi-bin/min2cgi.py"
   method="POST">
<p>Navn:
<textarea name="navn" value="1" cols="20" rows="1">
</textarea></p>

<p>Adresse:
<textarea name="adresse" value="2" cols="20" rows="1">
</textarea></p>

<input type="submit" value="Send inn">
</form>
</html>
```

Vi lager mottagerscriptet slik:

```
#! /usr/bin/python2

import cgi
form=cgi.FieldStorage()

print 'Content-type: text/plain\n'

print form
navn=''
adresse=''
try:
   navn=form['navn'].value
except:
   navn='ukjent'
try:
   adresse=form['adresse'].value
except:
   adresse='ukjent'
print 'Dette er det jeg fant:\n'
print 'Navn: '+navn+'\n'
print 'Adresse: '+adresse+'\n'
```

Vi ser at her har vi importert modulen cgi. Det gjøre det mulig for oss å få input dataene, parametrene, til scriptet ordnet i et FieldStorage som i all hovedsak oppfører seg som et Python dictionary (form=cgi.FieldStorage()). Det første vi gjør er å skrive denne rett ut. Etterpå sjekker vi hva vi faktisk har fått inn og lager en litt mer menneskevennlig utskrift.

Eksempel 3

Dette eksempelet gjør det samme som eksempel 2, bortsett fra at det produserer en full HTML-fil som returneres. Koden nedenfor er derfor et alternativ til scriptet over.

```
#! /usr/bin/python2

import cgi, string
print 'Content-type: text/html\n'

# preparing the content in body-element
form=cgi.FieldStorage()
navn=''
adresse=''
try:
   navn=form['navn'].value
except:
   navn='ukjent'
try:
   adresse=form['adresse'].value
except:
   adresse='ukjent'

# preparing the complete HTML-string
S=
"""<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<META http-equiv="Content-Type" content="text/html;
      charset=iso-8859-1\">
<title>navn og adresse</title>
</head>
<body>
<h2>Dette er det jeg fant</h2>
<p>Navn: %s</p>
<p>Adresse: %s</p>
</body>
</html>
"""
S=S%(navn,adresse)
print S
```

Eksempel 4

Dette eksempelet rapporterer alle omgivelsen til Python på tjeneren. Det er altså denne informasjonen vi kan benytte oss av når vi skriver CGI-skript, i tillegg til parametere og cookies.

```
#! /usr/bin/python2.4
print "Content-type: text/html"
print
print "<pre>"
import os, sys
from cgi import escape
print "<strong>Python %s</strong>" % sys.version
keys = os.environ.keys()
keys.sort()
for k in keys:
    print "%s\t%s" % (escape(k), escape(os.environ[k]))
print "</pre>"
```

DEMO: http://www.ia.hiof.no/~borres/cgi-bin/what/version.py

Referanser og eksempler:

- Detaljert om chmod()
  catcode.com/teachmod/
- Modulen: HTTP

DOM i Python

Hva:

DOM-programmering i Python, Minidom

Python har et apparat for å håndtere DOM. Det vil si at Python har definert et programmeringsgrensesnitt som lar oss etablere og operere på DOM-trær. Python har ikke implementert full DOM, men har implementert det de kaller minidom. For å få tilgang til full DOM-funksjonalitet i henhold til DOM spesifikasjonen fra W3C. må vi importere en slik implementasjon, f.eks. PyXML eller 4Suite. Vi konsentrerer oss om minidom i denne modulen.

Minidom i Python imlementerer en del av W3C's definisjon av programmeringsgrensesnittet mot DOM og løser de fleste av de praktiske oppgavene vi skal løse, selv om det noen ganger blir litt omstendelig.

Eksempel: Olympiade-data

Vi tar for oss resultatfila fra olympiade-eksempelet, se modulene: Olympiade og Noen datasett. De aktuelle resultatene er ordnet i en XML-fil: http://www.ia.hiof.no/~borres/commondata/olympiade/all_results_xml.html

Vi skal gjøre to øvelser på denne fila

1. Produsere en HTML-fil. Dette er i prinsipp samme transformasjon som den som gjøres ved XSLT i modulen: XML-->HTML
2. Søke etter en bestemt deltager i alle øvelser i begge olymiader.

Øvelse 1

Vi tar utgangspunkt i følgende Python program:

```
import xml.dom.minidom
import string

#--------------------------------------------------------------
# Simple demo of dom.
# produce rudimetary html from xml-file with IOC-results
# B. Stenseth  2002
# Use:
# DoIt(infile,outfile)
# See default files below
#--------------------------------------------------------------

# file io
def getTextFile(filename):
    try:
        file=open(filename,'r')
        intext=file.read()
        file.close()
        return intext
    except:
        print 'Error reading file ',filename
        return ''

def storeTextFile(filename,txt):
    try:
        outfile=open(filename,'w')
        outfile.write(txt)
        outfile.close()
    except:
        print 'Error writing file ',filename

eol='\n'

# collect all text in a node
def getText(nodelist):
    rc = ''
    for node in nodelist:
        if node.nodeType == node.TEXT_NODE:
            t=node.data.encode('ISO-8859-1')
            rc += t
    return rc

HTMLFile=
"""<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
  <META http-equiv="Content-Type" content="text/html;
        charset=iso-8859-1\">
  <title>Olympiade</title>
</head>

<body>

%s

</body>
</html>
"""


def handleIOC(doc):
    S=''
    games=doc.getElementsByTagName("OlympicGame")
    for game in games:
```

```
        S+=handleGame(game)
        S+=eol
    return S

def handleGame(game):
    S= '<h2>%s</h2>\n' %game.getAttribute('place').encode('ISO-8859-1')
    events=game.getElementsByTagName("event")
    for event in events:
        S+=handleEvent(event)
        S+=eol
    return S

def handleEvent(event):
    S= '<h3>%s</h3>\n' %event.getAttribute('track').encode('ISO-8859-1')
    participants=event.getElementsByTagName("athlet")
    for athlet in participants:
        S+=handleAthlet(athlet)
        S+=eol
    return S

def handleAthlet(athlet):
    name=athlet.getElementsByTagName("name")[0]
    S= "<p>Name:%s<br/>" %getText(name.childNodes)
    result=athlet.getElementsByTagName("result")[0]
    S+= "Result:%s</p>" %getText(result.childNodes)
    return S

# default file for demopurposes, change it
def doit(infile='c:\\articles\\ml\\dompython\\all_results.xml',
         outfile='c:\\articles\\ml\\dompython\\py_results1.html'):

    document=getTextFile(infile)

    dom = xml.dom.minidom.parseString(document)

    T=handleIOC(dom)
    outText=HTMLFile%T

    storeTextFile(outfile,outText)


    # clean up
    dom.unlink()
```

Programmet foretar en enkel transformasjon av en xml-struktur til en rudimentær html-string. Sammenlign denne koden med en tilsvarende XSLT-transformasjon som er beskrevet i Olympiade-eksempelet:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml"
     version="1.0"
     encoding="ISO-8859-1"
     indent="yes"
     doctype-public="-//W3C//DTD XHTML 1.0 Strict//EN"
     doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"/>
<xsl:template match="/">
    <html>
      <head>
        <link rel="STYLESHEET" href="olymp5.css"/>
         <title>Olympics</title>
       </head>
       <body>
```

```
          <h1>Resultater sprint</h1>
          <hr/>
          <xsl:apply-templates select="IOC/OlympicGame">
          <xsl:sort select="@year" order="ascending"/>
          </xsl:apply-templates>
         </body>
    </html>
</xsl:template>

<xsl:template match="OlympicGame">
  <table cellpadding="10">
    <tr>
      <td>
      <xsl:element name="img">
          <xsl:attribute name="src">
            <xsl:value-of select="@place"/>.gif</xsl:attribute>
          <xsl:attribute name="alt">
             <xsl:value-of select="@place"/></xsl:attribute>
      </xsl:element>
         </td>
         <td>
         <h1><xsl:value-of select="@place"/> <br/>
         <xsl:value-of select="@year"/></h1>
         </td>
      </tr>
  </table>

  <table cellpadding="10" border="0" cellspacing="0">
    <tr>
      <xsl:apply-templates select="event"/>
    </tr>
  </table>
</xsl:template>

<xsl:template match="//event">
  <td valign="top">
    <h2><xsl:value-of select="@track"/></h2>
     <xsl:apply-templates select="athlet">
     <xsl:sort  data-type="number" select="result"/>
     </xsl:apply-templates>
  </td>
</xsl:template>

<xsl:template match="athlet">
   <p><xsl:value-of select="name"/><br/>
   <xsl:value-of select="nation"/><br/>
   <xsl:value-of select="result"/></p>
</xsl:template>
</xsl:stylesheet>
```

DEMO: http://www.ia.hiof.no/~borres/ml/dompython/py_results1.html

Øvelse 2

Vi skriver et program som tar for seg våre olympiske data og forsøker å besvare spørsmålet: "I hvilke øvelser har nn deltatt i de aktuelle olympiadene". Dette innebærer at vi må gå ned og opp i treet. Først må vi lokalisere alle forekomstene av den aktuelle løperen, for deretter å gå opp i treet for å finne øvelse og olympiade.

```
import xml.dom.minidom

#--------------------------------------------------------------
# Simple demo of dom.
# find: report which events an athlet has participated in
# B. Stenseth  2002
# Use:
# Find(athlet,file)
# See default parametes below
#--------------------------------------------------------------
# file io
def getTextFile(filename):
    try:
        file=open(filename,'r')
        intext=file.read()
        file.close()
        return intext
    except:
        print 'Error reading file ',filename
        return ''

# collect all text in a node
def getText(nodelist):
    rc = ''
    for node in nodelist:
        if node.nodeType == node.TEXT_NODE:
            t=node.data.encode('ISO-8859-1')
            rc += t
    return rc

def searchIOC(doc,theName):
    athletnamelist=doc.getElementsByTagName("name")
    for athletname in athletnamelist:
        txtname=getText(athletname.childNodes)
        if txtname==theName:
            event=athletname.parentNode.parentNode
            game=event.parentNode
            print game.getAttribute('place').encode('ISO-8859-1')
            print ' - '+event.getAttribute('track').encode('ISO-8859-1')

# default parameters for demopurposes
def find(runner='Frank Fredericks',
         afile='c:\\articles\\ml\\olymp\\all_results.xml'):
    document=getTextFile(afile)

    dom = xml.dom.minidom.parseString(document)
    searchIOC(dom,runner)
```

Eksempel: Bok-data

Datagrunnlaget er en tekstfil med bokbeskrivelser, en bok på hver linje. Bokdataene er beskrevet i modulen Noen datasett. Bokliste som tekst http://www.ia.hiof.no/~borres/commondata/bokliste/bokliste.txt . Tomme linjer og linjer som begynner med // skal ignoreres.

Vi skal gjøre to øvelser på disse dataene:

1. Bygge en XML-fil fra textfila (csv-fila)
2. Endre strukturen på den fila vi bygger i øvelse 1.

Øvelse 1

Vi lager et Pythonprogram som tar for seg en tekstfil med bokbeskrivelser og lager en XML-fil.

```
import string,StringIO,xml.dom.minidom,codecs

#--------------------------------------------------------------
# Demo of MINIDOM.
# Building a DOM-tree based on a text-file, writing result as XML
# Building each node and inserting it into the tree
# Data is described on
# http://www.ia.hiof.no/~borres/ml/python/p-python.html
#
# Useage: doit(textfilename,xmlfilename)
# See below for default filenames
# B. Stenseth  2002
#--------------------------------------------------------------

# file io
def getTextFile(filename):
    try:
        file=open(filename,'r')
        intext=file.read()
        file.close()
        return intext
    except:
        print 'Error reading file ',filename
        return ''

def storeTextFile(filename,txt):
    try:
        outfile=open(filename,'w')
        outfile.write(txt)
        outfile.close()
    except:
        print 'Error writing file ',filename


def doit(infile='c:\\articles\\ml\\python\\bokliste.txt',
         outfile='c:\\articles\\ml\\dompython\\bokliste2.xml'):
    txt=getTextFile(infile)
    if(txt==''):
    return
    # prepare this string for unicode in a domtree
    txt=txt.decode('ISO-8859-1')
    lines=string.split(txt,'\n')

    # set up basic document
    doc=xml.dom.minidom.Document()
    root_elt=doc.createElement('booklist')
    doc.appendChild(root_elt)

    # walk the linelist
    linecount=0
    for line in lines:
        line=string.strip(line)
        # skip the blanks and the comments
        if len(line) <3:
            continue
        if line[0:2]=="//":
            continue
        # we will use it
        # title,author,publisher,year,isbn,pages,course,category,comment
        pieces=string.split(line,',');
        if len(pieces)!=9:
            # bad line
            print "ignore: " + line
            continue
```

```
        # make book
    book_elt_node=doc.createElement('book')
    book_elt_node.setAttribute('isbn',pieces[4])
    book_elt_node.setAttribute('pages',pieces[5])
    root_elt.appendChild(book_elt_node)

    new_elt_node=doc.createElement('title')
    new_elt_node.appendChild(doc.createTextNode(pieces[0]))
    book_elt_node.appendChild(new_elt_node)

    new_elt_node=doc.createElement('course')
    new_elt_node.appendChild(doc.createTextNode(pieces[6]))
    book_elt_node.appendChild(new_elt_node)

    new_elt_node=doc.createElement('category')
    new_elt_node.appendChild(doc.createTextNode(pieces[7]))
    book_elt_node.appendChild(new_elt_node)

    new_elt_node=doc.createElement('author')
    new_elt_node.appendChild(doc.createTextNode(pieces[1]))
    book_elt_node.appendChild(new_elt_node)

    new_elt_node=doc.createElement('publisher')
    new_elt_node.appendChild(doc.createTextNode(pieces[2]))
    book_elt_node.appendChild(new_elt_node)

    new_elt_node=doc.createElement('year')
    new_elt_node.appendChild(doc.createTextNode(pieces[3]))
    book_elt_node.appendChild(new_elt_node)

    new_elt_node=doc.createElement('comment')
    new_elt_node.appendChild(doc.createTextNode(pieces[8]))
    book_elt_node.appendChild(new_elt_node)


    # raw print while testing
    # print doc.toxml().encode('ISO-8859-1')

    # get it on file
    # need the domtree, doc, as a ISO-8859-1 encoded string

    s=StringIO.StringIO()
    doc.writexml(codecs.getwriter('ISO-8859-1')(s))
    # some dirty formatting, take care
    s=string.replace(s.getvalue(),'>','>\n')
    s=string.replace(s,'<book','\n\n<book')
    # fix prolog
    prolog="""<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE booklist SYSTEM "bokdok.dtd">"""
    s=string.replace(s,'<?xml version="1.0" ?>',prolog)
    # while testing
    print s

    storeTextFile(outfile,s)
    doc.unlink()
```

Dette gjøres ved å bygge opp et DOM-tre og ved å sette inn noder som genereres fra teksten. Denne Pythonkoden gjør i prinsipp det samme som koden som er beskrevet i modulen: Py-Eksempler. Der beskrives et preogram som gjør det samme som ren tekstbehandling, uten bruk av DOM,

DEMO: http://www.ia.hiof.no/~borres/ml/dompython/bokliste2.xml

Øvelse 2

Vi lager et program som tar for seg en XML-fil som bygget i øvelse 1 og endrer strukturen på denne, et element gjøres om til attributt og en attributt gjøres om til element.

```
import string,StringIO,codecs,xml.dom.minidom

#--------------------------------------------------------------
# Demo of MINIDOM.
# Changing the structure of a XML-file
# Data is described on
# http://www.ia.hiof.no/~borres/ml/python/p-python.html
#
# change it to make:
# all titles an attribute in stead of an element
# all pages an element in stead of an attribute
# B. Stenseth  2002
# Use: doit(infile,outfile)
# See below for default filenames
#--------------------------------------------------------------
# file io
def getTextFile(filename):
    try:
        file=open(filename,'r')
        intext=file.read()
        file.close()
        return intext
    except:
        print 'Error reading file ',filename
        return ''

def storeTextFile(filename,txt):
    try:
        outfile=open(filename,'w')
        outfile.write(txt)
        outfile.close()
    except:
        print 'Error writing file ',filename

# collect all text in a node
def getText(nodelist):
    rc = ''
    for node in nodelist:
        if node.nodeType == node.TEXT_NODE:
            t=node.data.encode('ISO-8859-1')
            rc += t
    return rc

def getStrippedText(nodelist):
    rc = ''
    for node in nodelist:
        if node.nodeType == node.TEXT_NODE:
            t=node.data
            t=string.strip(t)
            t=node.data.encode('ISO-8859-1')
            if t!='\n':
                rc += string.strip(t)
    return rc


def doit(infile='c:\\articles\\ml\\dompython\\bokliste2.xml',
         outfile='c:\\articles\\ml\\dompython\\bokliste3.xml'):
    txt=getTextFile(infile)
    if(txt==''):
    return
    # prepare this string for unicode in a domtree
    # txt=txt.decode('ISO-8859-1')
    doc = xml.dom.minidom.parseString(txt)
```

```
books=doc.getElementsByTagName('book')
for book in books:
    # pick up the title-element
    title_elt=book.getElementsByTagName('title')[0]
    title_str=getStrippedText(title_elt.childNodes)
    # make the title an attribute
    book.setAttribute('title',title_str.decode('ISO-8859-1'))
    # remove the title element
    book.removeChild(title_elt)

    # pick up the pages-attribute
    page_str=book.getAttribute('pages')
    # make the element
    page_elt=doc.createElement('pages')
    # make the text child node
page_elt.appendChild(doc.createTextNode(page_str))
book.appendChild(page_elt)
# remove pages-attribute
book.removeAttribute('pages')


# get it on file
# need the domtree, doc, as a ISO-8859-1 encoded string

s=StringIO.StringIO()
doc.writexml(codecs.getwriter('ISO-8859-1')(s))
s=s.getvalue()
# fix prolog
prolog='<?xml version="1.0" encoding="ISO-8859-1" ?>'
s=string.replace(s,'<?xml version="1.0" ?>',prolog)
s=string.replace(s,'bokdok.dtd','bokdok2.dtd')
# while testing
# print s

storeTextFile(outfile,s)
doc.unlink()
```

DEMO: http://www.ia.hiof.no/~borres/ml/dompython/bokliste3.xml

Eksempel: Skøyte-data

Tema er skøyteløp med egne tekstfiler som angir resultater fra 500m, 1500m, 5000m og 10000m. Disse tekstfilene er svært enkle og inneholder ett navn og ett resultat på hver linje. Filene heter henholdsvis s500.txt, s1500.txt, s5000.txt, s10000.txt. Vi skriver et program som gjør følgende:

1. Leser de fire filene og etablerer et DOM-tre for hver av dem
2. Slår sammen de fire trærne til ett
3. Beregner samlet poengsum for hver løper
4. Sorterer alle løpernodene etter beregnet resultat
5. Lager en HTML-fil der løperne vises sortert på resultat

Dette er neppe noe optimal måte å løse problemet på, men kan tjene som en DOM-øvelse. De 5 stegene er markert i Pythonkoden.

```
import string,StringIO,xml.dom.minidom,codecs

#--------------------------------------------------------------
# Demo of MINIDOM.
# NOTE that this may not be the smartest or fastest way to
# solve this problem. It is written to demonstrate minidom
#
# function makeCompleteXML(catalog)
# Building a XML-file based on three text-files:
# Results from 500, 1500, 5000, 10000 m speedskating
# each with lines of the form(not sorted):
# name,result
# Filenames are s500.txt, s1500.txt, s5000.txt, s10000.txt
# Returns a tree with following structure:
#
#<?xml version="1.0" encoding="ISO-8859-1" ?>
#<skatingevent>
#<skater name="olsen">
#   <res500>40.00</res500>
#   <res1500>1.50.00</res1500>
#   <res5000>6.40.00</res5000>
#   <res10000>13.40.00</res10000>
#   <points>87559</points>
#</skater>
# ...
#</skatingevent>
#
# Function doit(catalog)
# calls storeXMLFile and produce a sorted html-file:
  skaters.html
# Job is done in 4 commented steps:
#   Read the 4 txtfiles and establish a DOM-tree for each
#   Joins the 4 trees to one tree
#   Calculates aggregated points for each skater
#   Sort skaters on points
#   Make an HTML-file of sorted skaters
#
# Usage: doit(catalog)
#
# B. Stenseth  2003
#--------------------------------------------

def getText(nodelist):
    # collect all text in a node
    rc = ''
    for node in nodelist:
        if node.nodeType == node.TEXT_NODE:
            t=node.data.encode('ISO-8859-1')
            rc += t
    return rc


def makeTree(catalog,distanse):
    # read a file from the catalog and establish tree
    try:
    filename=catalog+'\\s'+distanse+'.txt'
    # sample: c:\myskatingfiles\s500.txt
    file=open(filename,'r')
    intxt=file.read()
    file.close()
    intxt=intxt.decode('ISO-8859-1')

        doc=xml.dom.minidom.Document()
        root_elt=doc.createElement('skatingevent')
```

```
        doc.appendChild(root_elt)
    lines=string.split(intxt,'\n')
    for line in lines:
            pieces=string.split(line,',')
            if len(pieces)==2:
                skater_elt=doc.createElement('skater')
                skater_elt.setAttribute('name',pieces[0])
                result_elt=doc.createElement('res'+distanse)
                result_elt.appendChild(doc.createTextNode(pieces[1]))
                skater_elt.appendChild(result_elt)
                root_elt.appendChild(skater_elt)
    return doc
    except:
        print 'Error building: '+distanse
        return ''


def storeXMLFile(filename,doc):
    # storing an xmlfile from a tree
    s=StringIO.StringIO()
    doc.writexml(codecs.getwriter('ISO-8859-1')(s))
    t=s.getvalue()
    # some dirty formatting, take care
    t=string.replace(t,'<skater','\n<skater')
    t=string.replace(t,'<res','\n<res')
    # fix prolog
    prolog='<?xml version="1.0" encoding="ISO-8859-1" ?>'
    t=string.replace(t,'<?xml version="1.0" ?>',prolog)
    # print while storing if you want to test
    # print t
    try:
        outf=open(filename,'w')
        outf.write(t)
        outf.close()
    except:
        print 'Error in writing tree at:'+ filename


def makeCompleteXML(catalog='c:\\articles\\ml\\dom'):
    # produce the complete tree with results from all distances
    # strategy is to make a tree for each distance and then join them
    # make a tree for each distance

    #--------------------------------------
    # STEP 1 make 4 DOM-trees
    t500=makeTree(catalog,'500')
    t1500=makeTree(catalog,'1500')
    t5000=makeTree(catalog,'5000')
    t10000=makeTree(catalog,'10000')
    #--------------------------------------

    # store them and print them while testing
    #storeXMLFile(catalog+'\\xml500.xml',t500)
    #storeXMLFile(catalog+'\\xml1500.xml',t1500)
    #storeXMLFile(catalog+'\\xml5000.xml',t5000)
    #storeXMLFile(catalog+'\\xml10000.xml',t10000)

    #--------------------------------------
    # STEP 2 join trees to one tree
    # use t500 as master and assemble results from the three others
    d500=t500.getElementsByTagName('skater')
    d1500=t1500.getElementsByTagName('skater')
    d5000=t5000.getElementsByTagName('skater')
    d10000=t10000.getElementsByTagName('skater')
```

```
    for p500 in d500:
        name500=p500.getAttribute('name')
        for p1500 in d1500:
            if name500==p1500.getAttribute('name'):
                p500.appendChild(p1500.getElementsByTagName('res1500')[0] )
                break
        for p5000 in d5000:
            if name500==p5000.getAttribute('name'):
                p500.appendChild(p5000.getElementsByTagName('res5000')[0] )
                break
        for p10000 in d10000:
            if name500==p10000.getAttribute('name'):
                p500.appendChild(p10000.getElementsByTagName('res10000')[0] )
                break
    #--------------------------------------

    # now we have all results in t500
    # write it if you want
    # storeXMLFile(catalog+'\\sall.xml',t500)

    #--------------------------------------
    # STEP 3 calculated aggregated points for each skater
    # we want to calculate points for each skater and add
    # an element points to each skater
    skaters=t500.getElementsByTagName('skater')
    for skater in skaters:
        # calculate timepoints in 1/100 seconds
        s=getText(skater.getElementsByTagName('res500')[0].childNodes)
        hsecs=makeSeconds(s)
        s=getText(skater.getElementsByTagName('res1500')[0].childNodes)
        hsecs+=makeSeconds(s)/3.0
        s=getText(skater.getElementsByTagName('res5000')[0].childNodes)
        hsecs+=makeSeconds(s)/10.0
        s=getText(skater.getElementsByTagName('res10000')[0].childNodes)
        hsecs+=makeSeconds(s)/20.0
        points='%.3f' %(hsecs/100.0)

        point_elt=t500.createElement('points')
        skater.appendChild(point_elt)
        point_elt.appendChild(t500.createTextNode(points))
    #--------------------------------------

    # and you may save it again while testing
    # storeXMLFile(catalog+'\\sallpoints.xml',t500)

    # clean up
    t1500.unlink()
    t5000.unlink()
    t10000.unlink()

    return t500

def makeSeconds(s):
    # calculate 1/100 seconds from s
    # s in form mm.ss.hh ( minutes, seconds, 1/100 seconds)
    # print s
    parts=string.split(s,'.')
    hsecs=0
    if len(parts)==3:
        hsecs=6000*int(parts[0])+100*int(parts[1])+int(parts[2])
    elif len(parts)==2:
        hsecs=100*int(parts[0])+int(parts[1])
    else:
        print 'error in timeformat: ' + s
```

```
        hsecs=9999999
    return hsecs


def compareSkaters(s1,s2):
    # used while sorting
    s1pnt=s1.getElementsByTagName('points')[0]
    s2pnt=s2.getElementsByTagName('points')[0]
    v1=int(float(getText(s1pnt.childNodes)))
    v2=int(float(getText(s2pnt.childNodes)))
    return v1 - v2

def doit(catalog='c:\\articles\\ml\\dompython'):
    # make the complete job from 4 text-files to html-file
    # first we build the complete xml-tree
    # including points and all results and calculated points
    #--------------------------------------
    # STEPS 1,2,3 as commented on top of script
    doc=makeCompleteXML(catalog)
    #--------------------------------------

    # and we may save it just to test
    storeXMLFile(catalog+'\\sallpoints.xml',doc)

    #--------------------------------------
    # STEP 4 sort skaters according to calculated points
    # we want to sort on points
    skaters=doc.getElementsByTagName('skater')
    skaters.sort(compareSkaters)
    #--------------------------------------

    #--------------------------------------
    # STEP 5 produce a HTML-page
    # now we want to produce some html-output with results
    T="""<html>
    <head> <title>resultater</title>
    <body>
    <h1 style="font-size:14px">Resultater</h1>
    """
    # run through sorted list of skaters
    T+='<table cellpadding="2">\n'
    for skater in skaters:
        T+='<tr><td style="font-size:12px">'
        T+=skater.getAttribute('name').encode('ISO-8859-1')
        T+='</td><td style="font-size:12px">'
        T+=getText(skater.getElementsByTagName('points')[0].childNodes)
        T+='</td></tr>\n'

    T+='</tr>\n</table>\n</body>\n</html>\n'
    filename=catalog+'\\skaters.html'
    outf=open(filename,'w')
    outf.write(T)
    outf.close()
    #--------------------------------------

    doc.unlink()
```

DEMO: http://www.ia.hiof.no/~borres/ml/dompython/skaters.html

Referanser og eksempler:

DOM i Python:

- XML for Python pyxml.sourceforge.net/
- 4Suite: 4suite.org/index.xhtml

Python kode og rådata er sitert i teksten, unntatt skøyteresultatene: skoytefiler.zip

SAX i Python

Hva:

SAX-programmering i Python

Python har et apparat for SAX. Siden SAX er en eventdrevet mekanisme, finner vi en implementasjon som ligner den vi finner i andre eventdrevne omgivelser. Python har en parser som leser en string og detekterer de substringene som skal gi opphav til et event. Det er vår oppgave å fortelle Pythonparseren hva som skal skje når et event intreffer. Dette gjør vi ved å angi callback-funksjoner. Det vil si at vi forteller parseren hvilke av våre egendefinerte funksjoner som skal kalles ved de respektive begivenhetene. Dette er en strategi som er helt parallell til den vi bruker når vi lager programmer som er styrt av brukerbegivenheter. Vi skriver da metoder for å plukke opp museklikk osv. Her ligger det også en "parser" i bakgrunnen som sender en melding eller kaller opp vår funksjon når det inntreffer en begivenhet som vi har meldt vår interesse i. I en SAX-parsing er ikke begivenhetene museklikk men symboler i den stringen som analyseres.

Vi bruker resultatfila fra olympiade-eksempelet, se modulene: Olympiade og Noen datasett. De aktuelle resultatene er ordnet i en XML-fil: all_results.xml

Eksempel 1

```
import xml.parsers.expat

#--------------------------------------------------------------
# Simple demo of sax.
# B. Stenseth  2002
# Use:
# Show(filename)
# See below for default filename
#--------------------------------------------------------------

# helper to load a text file
def getTextFile(filename):
    try:
        file=open(filename,'r')
        intext=file.read()
        file.close()
        return intext
    except:
        print 'Error reading file ',filename
        return ''

#-------------------------------------------------------
# 3 primitive handler functions used by show
def start_element(name, attrs):
    print 'Start:', name, attrs
def end_element(name):
    print 'End:', name
def char_data(data):
    #print 'Data:', repr(data)
    if(not data.isspace()):
        print 'Data:', data

# default filename for demopurposes, change it
def showAll(afile='c:\\pydemo\\all_results.xml'):
    p = xml.parsers.expat.ParserCreate()
    p.returns_unicode = 0
    p.StartElementHandler = start_element
    p.EndElementHandler = end_element
    p.CharacterDataHandler = char_data

    content=getTextFile(afile)

    p.Parse(content)

def showData(afile='c:\\pydemo\\all_results.xml'):
    p = xml.parsers.expat.ParserCreate()
    p.returns_unicode = 0
    p.CharacterDataHandler = char_data

    content=getTextFile(afile)

    p.Parse(content)


showData()
```

Det er disse linjene som setter opp koplingen:

```
p.StartElementHandler = start_element
p.EndElementHandler = end_element
p.CharacterDataHandler = char_data
```

der start_element, end_element og char_data er våre funksjoner. Se Python dokumentasjonen for en komplett liste av hvilke events vi kan plukke opp.

Eksempel 2

Nedenfor finner du et eksempel som plukker ut og rapporterer verdien til alle elementer med tagnavn "name".

```
import xml.parsers.expat,string

#-------------------------------------------------------------
# Simple demo of sax. demosax2
# selects content of element: name
# B. Stenseth  2002
# Use:
# Show(filename)
# See below for default filename
#-------------------------------------------------------------

# helper to load a text file
def getTextFile(filename):
    try:
        file=open(filename,'r')
        intext=file.read()
        file.close()
        return intext
    except:
        print 'Error reading file ',filename
        return ''

# global flag
navnFlag=-1
#-----------------------------------------------------
# 3 primitive handler functions used by show
def start_element(name, attrs):
    global navnFlag
    navnFlag= string.find(name,'name')

def end_element(name):
    global navnFlag
    navnFlag=-1

def char_data(data):
    global navnFlag
    if navnFlag>=0:
        print data

# default filename for demopurposes, change it
def show(afile='c:\\pydemo\\all_results.xml'):
    p = xml.parsers.expat.ParserCreate()
    p.returns_unicode = 0
    p.StartElementHandler = start_element
    p.EndElementHandler = end_element
    p.CharacterDataHandler = char_data

    content=getTextFile(afile)

    p.Parse(content)
```

Eksempel 3

Nedenfor finner du et eksempel som skriver en indentert utskrift av alle olympiader, øvelser og deltagere.

```
import xml.parsers.expat,string

#-------------------------------------------------------------
# Simple demo of sax. demosax2
# selects content of element: name
# B. Stenseth  2006
# Use:
# Show(filename)
# See below for default filename
#-------------------------------------------------------------

# helper to load a text file
def getTextFile(filename):
    try:
        file=open(filename,'r')
        intext=file.read()
        file.close()
        return intext
    except:
        print 'Error reading file ',filename
        return ''

# global flag
navnFlag=-1
# indentation  when writing
indent='    '
#-----------------------------------------------------
# 3 primitive handler functions used by show
def start_element(name, attrs):
    global navnFlag
    navnFlag= string.find(name,'name')
    if name.find('OlympicGame')!=-1:
            print attrs['place']
    if name.find('event')!=-1:
            print indent+attrs['track']

def end_element(name):
    global navnFlag
    navnFlag=-1

def char_data(data):
    global navnFlag
    if navnFlag>=0:
        print indent+indent+data

# default filename for demopurposes, change it
def show(afile='c:\\pydemo\\all_results.xml'):
    p = xml.parsers.expat.ParserCreate()
    p.returns_unicode = 0
    p.StartElementHandler = start_element
    p.EndElementHandler = end_element
    p.CharacterDataHandler = char_data
    content=getTextFile(afile)
    p.Parse(content)

show()
```

Referanser og eksempler:

Akuelle filer er sitert i sin helhet i koden

Objekter i Python

hvordan lage egne klasser og objekter

Hva:

Litt om egne klasser og objekter i Python

Moduler og klasser

Python opererer med moduler og objekter.

Moduler

Modulbegrepet er trolig inspirert av Pascal eller Modula. Vi vet at vi kan lage moduler med funksjoner og aksessere disse funksjonene ved hjelp av dot-notasjon. F.eks. slik:

```
import math
print math.sqrt(25)
```

Vi importerer modulen math og bruker funksjonen sqrt() som er definert i math. math er ikke en klasse, men en modul. I dette tilfellet brukte vi en standardmodul, men vi vet at vi kan lage våre egne moduler. Vi vet videre at vi kan aksessere, ikke bare funksjoner, men også globale variable i moduler. F.eks. slik:

```
import math
print math.sin(math.pi/2.0)
```

Det greieste er vel å betrakte moduler som en måte å ordne biblioteker på, selv om det i programkoden kan se ut som vi opererer mot objekter. Av og til har vi klasser i moduler. F.eks:

```
import datetime
now=datetime.date.today()
print now.year
mydate=datetime.date(1946,12,27)
print mydate.day
```

Vi bruker klassen date i modulen datetime. Vi bruker først den statiske funksjonen today() for å konstruere et date-objekt. Deretter lager vi et date-bjekt ved hjelp av en konstuktor som tar år, mnd, dag som input.

Klasser

Python er et objektorientert språk. Vi kan betrakte datatyper og innebygde datastrukturer som objekter.

Klasse begrepet i Python har endret seg. I Pythondokumentasjonen (language reference 3.3) uttrykkes dette slik: "Classes and instances come in two flavors: old-style or classic, and new-style."

Nedenfor skal vi se på noen svært enkle eksempler på egne klasser. Vi skal uttrykke oss i "new-style", men ikke gå spesielt dypt i materien. innføringen er basert på noen enkle eksempler.

Eksempel 1

I det første eksempelet lager vi en klasse for å beskrive personobjekter, og vi lager en person. Vi lager personklassen som en klasse som arver fra **object**. **objekt** er "alle klassers mor" i det nye klassebegrepet i Python. **object** har, i Python som i andre språk med tilsvarende kosntruksjon, noen grunnleggende metoder som er nyttig i en del sammenhenger. Vi forfølger ikke dette i det første eksempelet.

```
class person(object):
    def __init__(self,n,a):
        self.name=n
        self.address=a

ole=person('ole','Halden')
print ole.name
```

Merk at **__init__** fungerer som en konstruktor. Merk videre bruken av **self**. Siden Python ikke har noen mekanismer for å deklarere variable, brukes f.eks. self.address for å definere en lokal variabel address, "dette objektets address". **self** må alltid være i den formelle parameterlista i klassemetoder. Vi kan skrive om personklassen slik at den ligner på den formen vi vanligvis bruker i Javabeans.

```
class person(object):
    def __init__(self,n,a):
        self.name=n
        self.address=a
    def getName(self):
        return self.name
    def getAddress(self):
        return self.address
    def setName(self,n):
        self.name=n
    def setAddress(self,a):
        self.address=a
    def __str__(self):
        return self.name+' fra '+self.address

ole=person('ole','Halden')
print ole.__str__()
ole.setName('Ole Jakob')
print ole.__str__()
```

Merk at **__str__(self)** er en redefinisjon av tilsvarende funksjon i **object**.

Eksempel 2

Nå bygger vi ut personklassen, og definerer en subklasse: ansatt'.

```
class person(object):
    def __init__(self,n,a):
        self.name=n
        self.address=a
    def getName(self):
        return self.name
    def getAddress(self):
        return self.address
    def setName(self,n):
        self.name=n
    def setAddress(self,a):
        self.address=a
    def __str__(self):
        return self.name+' fra '+self.address

class ansatt(person):
    def __init__(self,n,a,j):
        person.__init__(self,n,a)
        self.job=j
    def getJob():
        return self.job
    def setJob(self,j):
        self.job=j
    def __str__(self):
        return person.__str__(self)+' har jobb som '+self.job

hans=ansatt('hans','Halden','vaktmester')
print hans.__str__()
hans.setName('Hans Jakob')
hans.setJob('regnskapssjef')
print hans.__str__()
print super(ansatt,hans).__str__()
```

med utskriften:

```
hans fra Halden har jobb som vaktmester
Hans Jakob fra Halden har jobb som regnskapssjef
Hans Jakob fra Halden
```

Eksempel 3

Et litt mer praktisk eksempel: Vi leser data fra en kommaseparert fil og bygger opp en liste av objekter, ett for hver linje. Som datagrunnlag bruker vi landbeskrivelser fra geonames. Datasettet er beskrevet i modulen Noen datasett, og dataene er slik: land.txt

```
import urllib
"""
Using a class to administrate geodata
"""
#---------------------------------
# a class to hold : countryname,continent,area, capital
class country(object):
    def __init__(self,plist):
        self.countryName=plist[4]
        self.continent=plist[8]
        self.capital=plist[5]
        if plist[6]=='':
            self.area=0.0
        else:
            self.area=float(plist[6])
    def __str__(self):
        return self.countryName
    def printAllOfMe(self):
        return str(self.area)+'\t'+self.countryName+' - '+self.capital

#-------------------------------
# continets [isocode,fullname]
continents=[['EU','Europa'],['AS','Asia'],['NA', 'North America'],
            ['AN','Antarctica'],['SA','South America'],['OC','Oceania'],
            ['','No region']]

# ---------------------------------
# read from whereever you place the data
# pick a country from each line and produce an object list
def makeCountryList():
    try:
        allCountries=[]
        # quick and dirty:
        surl='http://www.ia.hiof.no/~borres/commondata/geonames/land.txt'
        T=urllib.urlopen(surl).read()
        lines=T.split('\n')
        for line in lines:
            parts=line.split('\t')
            if len(parts)!=12 or line.startswith('iso'):
                continue
            allCountries.append(country(parts))
        return allCountries
    except:
        return[]

#-------------------
# report a list pr continent
def reportContinent(countryList):
    result=''
    for c in continents:
        result+=c[1]+'\n'
        for cobj in countryList:
            if cobj.continent==c[0]:
                result+='\t'+cobj.__str__()+'\n'
    return result

#------------------
# report all countries sorted on area

# sort helper
def compareCountries(c1,c2):
    if c1.area < c2.area:
        return 1
    if c2.area < c1.area:
```

```
        return -1
    return 0

def reportOnSize(countryList):
    result=''
    countryList.sort(compareCountries)
    for cobj in countryList:
        result+=cobj.printAllOfMe()+'\n'
    return result

#---------------------------
# testing the module
if __name__=="__main__":
    clist=makeCountryList()
    print reportOnSize(clist)
    print reportContinent(clist)
```

Det er en enkel sak å skrive om rapporteringen slik at vi får ut f.eks. et HTML-format. Hvis vi gjør dette og preparerer koden som et CGI-skript, få vi følgende:

-

```
#! /usr/bin/python2
import urllib
import cgi
import cgitb; cgitb.enable()
"""
Using a class to administrate geodata
"""
#---------------------------------
# a class to hold : countryname,continent,area, capital
class country(object):
    def __init__(self,plist):
        self.countryName=plist[4]
        self.continent=plist[8]
        self.capital=plist[5]
        if plist[6]=='':
            self.area=0.0
        else:
            self.area=float(plist[6])
    def __str__(self):
        return self.countryName
    def printAllOfMe(self):
        return str(self.area)+'\t'+self.countryName+' - '+self.capital

#-------------------------------
# continets [isocode,fullname]
continents=[['EU','Europa'],['AS','Asia'],['NA', 'North America'],
            ['AN','Antarctica'],['SA','South America'],['OC','Oceania'],
            ['','No region']]
# a simple HTML-fragment for a table row
TRFragment="<tr><td>%s</td><td>%s</td><td>%s</td><td>%s</td></tr>"
# a simple HTML-fragment for a table
TABLEFragment='<table cellpadding="5px" cellspacing="0px">%s</table>'

# ---------------------------------
# read from whereever you place the data
# pick a country from each line and produce an object list
def makeCountryList():
    try:
        allCountries=[]
        # quick and dirty:
        surl='http://www.ia.hiof.no/~borres/commondata/geonames/land.txt'
        T=urllib.urlopen(surl).read()
        lines=T.split('\n')
        for line in lines:
            parts=line.split('\t')
            if len(parts)!=12 or line.startswith('iso'):
                continue
            allCountries.append(country(parts))
        return allCountries
    except:
        return[]

#-------------------
# report a list pr continent
def reportContinent(countryList):
    result=''
    for c in continents:
        result+=TRFragment%(c[1],'','','')+'\n'
        for cobj in countryList:
            if cobj.continent==c[0]:
                result+=TRFragment%('',
                                    cobj.countryName,
                                    cobj.capital,
                                    cobj.area)+'\n'
```

```
    return TABLEFragment%result

#------------------
# report all countries sorted on area

# sort helper
def compareCountries(c1,c2):
    if c1.area < c2.area:
        return 1
    if c2.area < c1.area:
        return -1
    return 0

def reportOnSize(countryList):
    result=''
    countryList.sort(compareCountries)
    for cobj in countryList:
        result+=TRFragment%(cobj.area,
                            cobj.countryName,
                            cobj.capital,
                            cobj.continent)+'\n'
    return TABLEFragment%result

#---------------------------
# expect to find: job
form=cgi.FieldStorage()
print 'Content-type: text/html\n'
job='area'
if form.has_key('job'):
    job=form['job'].value
clist=makeCountryList()
if job=='area':
    print reportOnSize(clist)
elif job=='continent':
    print reportContinent(clist)
else:
    print 'noe gikk galt, feil job'
```

DEMO: http://www.ia.hiof.no/~borres/ml/pyobj/visland.html

Referanser og eksempler:

Pythondocumentasjonen (verson 2.5) er ikke oppdatert på det nye objektbegrepet
Se derfor: www.python.org/doc/newstyle.html

Geonames: www.geonames.org/

Python og MySQL

etter bidrag fra kollega Edgar Boström

Hva:

Kort om installering og bruk av MySQL fra Python

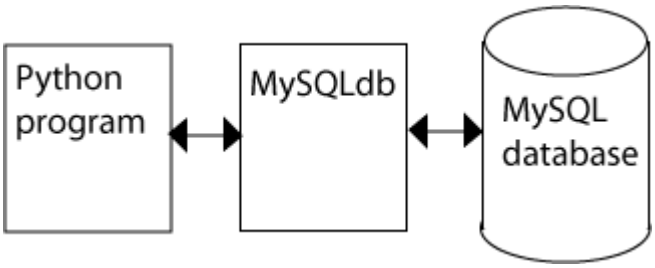
MySQL

- Last ned MySQL fra www.mysql.com og installer denne der du ønsker.
- Last evt. også ned "MySQL GUI Tools Bundle" fra samme sted. Her finner du verktøy for å administrere og inspisere databaser.
- Databasen må startes

Python bibliotek for MySQL

- Last ned MySQLdb via www.mysql.com, eller fra sourceforge.net/projects/mysql-python. Sjekk versjoner og kompatibilitet

Kjøring av SQL-statements i Python - prinsipp:



Når vi programmerer bruker vi MySQLdb-objekter for å koble oss opp, hente data osv. Dette blir dermed integrert i Python.

Opplegget blir i korthet:

- Koble opp databasen (lag et databaseoppkoblingsobjekt som peker til den maskinen og databasen du ønsker å jobbe med)
- Deklarer et resultatobjekt som kan eksekvere spørringer og hvor resultatet av spørringen mot databasen kan legges
- Bruk databasen
    - Legg spørringen i en tekststreng
    - Kjør spørringen
    - Bruk resultatet
- Koble ned databasen

Eksempel:

Edgar Bostrøm

```
Python 2.0 (#8, Oct 16 2000, 17:27:58) [MSC 32 bit (Intel)] on win32
Type "copyright", "credits" or "license" for more information.
IDLE 0.6 -- press F1 for help
>>> def MySQLdemo():

    # importer bibliotek, koble opp databasen
    # og sett opp en markør (resultatobjekt)
    # host, bruker, passord, database
    # som host kan du skrive ip, db er databasenavnet i MySQL
    import MySQLdb
    #forutsetter først at basen ligger lokalt, uten passord
    mindatabase=MySQLdb.connect(host='localhost',user='',passwd='',db='test')
    mintabell=mindatabase.cursor()

    # Legger inn en ny kunde. Gir ingen melding tilbake hvis alt er ok
    sql_setning="insert into kunde values (46,'Hansen &amp; Co')"
    mintabell.execute(sql_setning)

    #definer og eksekver en sporring.
    # NB: Hele SQL (også tabelloppretting) kan brukes.
    sql_setning = 'select * from kunde'
    mintabell.execute(sql_setning)

    # Lister ut de aktuelle, alt. 1, med feltnavn
    for kundenr,kundenavn in mintabell.fetchall():
    print 'Kunde --> ',kundenr,kundenavn
    print "ferdig1"

    # Lister ut de aktuelle, alt. 2, med feltnr
    for radene in mintabell.fetchall():
      print 'Kunde --> ',radene[0],radene[1]
    print "ferdig2"

    mindatabase.close()

  # eksempel som viser oppkobling mot ekstern database:
  # mindatabase =MySQLdb.connect
  # (host='pc16-81.hiof.no',user='iad10002',passwd='javaxerxgoy', db='skatt')
```

Resultatet av de to utlistings avsnittene ovenfor:

```
>>> MySQLdemo()
Kunde -->  1 Ole
Kunde -->  2 Hans
Kunde -->  46 Hansen & Co
ferdig1
Kunde -->  1 Ole
Kunde -->  2 Hans
Kunde -->  46 Hansen & Co
ferdig2
>>>
```

Referanser og eksempler:

- MySQL: www.mysql.com
- Kort om SQL: sqlzoo.net/
- MySQL for Python: sourceforge.net/projects/mysql-python

Noen eksempler på bruk av databaser i dette materialet:

- Fotballresultater: FIFA
- Enkel "blogg": IFA
- Fotball VM: FotballVM
- Olympiske data: XML-->SQL og DB-->XML

Ting og tang

Hva:

Ting og tang som er greitt å vite

endre path

Den typiske situasjonen er at vi ønsker å bruke en modul som ligger et sted som ikke er definert i den systemvariabelen som under Windows heter: PYTHONPAT. Måten å gjøre dette på er å endre pythonpath fra pythonkoden når vi trenger det. F.eks. kan det se slik ut:

```
import sys
sys.path+=['f:\\html\\cgi-bin\\mycat']
import mymodule
```

Du kan skrive ut stien til alle katalogene i sys.path slik:

```
for d in sys.path:
    print d
```

returnere en fil

I et CGI-script har vi ofte behov for å sende en ferdiglagd fil tilbake. Dette kan vi selvsagt gjøre ved å lese den inn i scriptet og skrive den tilbake på vanlig måte. Et alternative er følgende setning;

```
print "Location: http://www.ia.hiof.no/~borres/ml/histo/p-histo.html\n"
```

En slik print kommer som alternativ til den vanlige

```
print "Content-type: text/html\n"
print .....
```

tegnkode

Håndtering av ulike tegnkoder kan skape trøbbel for oss på mange måter. Ett av problemener er å få skrevet kommentarer og/eller tekstkonstanter med æ,ø,å i en kilde fil. Vi kan fortelle Python-interpretatoren hva slags koding den kan forvente å finne med linja:

```
# -*- coding: latin_1 -*-
```

F.eks. slik:

```
#! /usr/bin/python2
# -*- coding: latin_1 -*-
# i dette scriptet vil jeg prøve å ....
import random,string,time,cgi
```

Dette gjelder fra Python versjon 2.3. Mer om ulike kodemuligheter finner du i Pythondokuamenstasjonen, 4.9.2 Standard Encodings.

norske elementnavn

Vi vet at vi kan ha XML-filr som ser slik ut:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<liste>
<person>
    <navn>Bøler</navn>
    <kjønn>m</kjønn>
</person>
...
</liste>
```

Tekst noder med norske tegn (Bøler) kan vi handtere i minidom ved å sørge for at vi skriver noe slikt når vi henter teksten:

```
if node.nodeType == node.TEXT_NODE:
            t=node.data.encode('ISO-8859-1')
```

og slik hvis vi ønsker å legge noe til en DOM-struktur kan vi lage unicode ved å skrive:

```
nyttnavn=unicode(originalnavn,'ISO-8859-1')
```

Dersom vi ønsker å søke i et register etter elementer med navn som har norske tegn kan vi bruke samme mekanisme. Vi gjør altså søkekriteriet om til unicode før vi bruker det:

```
sexkey=unicode('kjønn','ISO-8859-1')
resultat=person.getElementsByTagName(sexkey)
```

andre programmer

Av og til har vi behov for å kjøre andre programmer eller kommandoer fra et pythonskript. Et eksempel er når vi vil kjøre FOP fra et skript på serveren. (mer om FOP i modulen: XSL-FO)

Generelt kan vi kjøre en kommando slik:

```
    os.system("kommando")
```

der kommando er det vil ville ha kjørt fra kommandolinja. F.eks. skal vi kunne handtere en FOP-operasjon og ta vare på resultatet av operasjonen slik:

```
    os.system("/usr/local/fop/fop.sh myfile.fo myfile.pdf")
    file=open('myfile.pdf','r')
    resultat=file.read()
    file.close()
```

Dersom vi har en kommando eller et program som skriver til standard output, kan vi fange opp resultatet slik:

```
    f=os.popen("kommando")
    resultat=f.read()
```

ÆØÅ

Python dokumentasjon beskriver hvordan Python kan handtere ulike typer karakterer (character encoding). Normalt går det bra når vi leser og skriver tekstfiler med norske tegn. Når vi imidlertid bruker dom eller minidom får vi problemer med våre norske tegn dersom vi ikke passer på. Python bruker unicode i dom og vi må kode om dette før vi kan bruke teksten i andre sammenhenger, og før vi skriver til fil eller std out.

Min erfaring er at det går bra når jeg koder om (encode) strengene jeg plukker fra dom-treet. Følgende rutine som plukker ut alt tekstmateriale i et element ser ut til å gjøre jobben:

```
# collect all text in a node
def getText(nodelist):
    rc = ''
    for node in nodelist:
        if node.nodeType == node.TEXT_NODE:
            t=node.data.encode('ISO-8859-1')
            rc += t
    return rc
```

En kodesekvens som bruker rutina kan være slik:

```
...
result=q.getElementsByTagName('headtxt')[0]
T=getText(result.childNodes)
...
```

Likeledes kan vi plukke opp en attributt slik:

```
...
T=en_node.getAttribute('att').encode('ISO-8859-1')
...
```

Vi kan bruke andre kodetyper enn ISO-8859-1, se Python dokumentasjonen. Begge eksemplene på stringkoding, tekstnoder og attributter, er brukt i eksemplene ovenfor.

laste ned en fil

Vi kan laste ned en fil fra nettet og lagre den som en lokal fil eller få tilgang til innholdet direkt.

```
import urllib
theUrl='http://www.ia.hiof.no/~borres/ml/python/frej1.txt'
theFile='c:\\tmp\\frej1.txt'

#Download a file and print it
try:
    f=urllib.urlopen(theUrl)
    S=f.read()
    f.close()
    print S
except:
    print 'cannot load'


#Download a file and save it locally
try:
    urllib.urlretrieve( theUrl,theFile)
except:
    print 'cannot copy'
```

traversere en katalog

Vi kan løpe gjennom alle filer i en katalogstruktur på en enkel måte. Koden nedenfor gjør dette og skriver ut fullt filnavn for alle filer.

```
import os
   ...
def printAllFiles(catalog):
    for root, dirs, files in os.walk(catalog, topdown=False):
        for name in files:
            print os.path.join(root, name)
```

Enumeration

Når vi lagrer data i lister blir ofte koden tunglest når vi aksesserer data ved index. Vi kan enkelt definere navn på de ulike indeksene.

```
(navn,year,city)=[0,1,2]
p1=['ole',1990,'Halden']
p2=['jens',1971,'Sarpsborg']
print p1[navn]
print p2[city]
```

,forutsatt at ikke listenes størrelse ordning og størrelse endres

Referanser og eksempler:

Testing av Pythonkode

med PyUnit

Hva:

Hvordan teste Pythonkode

Vi skal holde oss i "unit"-tradisjonen og se på PYUnit for testing av Pythonkode. PyUnit er en del av Pythondistribusjonen fra versjon 2.1(?).

Pythondokumentasjonen er ganske klar og eksplisitt når det gjelder filosofien i PyUnit. Jeg gjentar ikke dette her, men konsentrerer meg om noen konkrete eksempler.

Eksempel 1

Utgangspunktet er følgende "kravspesifikasjon":

> Vi ønsker å skrive en modul med en funksjon som konverterer et naturlig tall, gitt som text, til en sekvens av tallord.

Vi skal implementere konverteringen i en modul som heter numbers, som metoden convert. På grunnlag av dette skriver vi en minimalistisk testmodule:

```
"""
testing function convert in module numbers
"""
# import the module we want to test
import numbers
# import the testunit
import unittest

class testconversion(unittest.TestCase):
    def testConvert(self):
        # test conversion
        self.assert_(numbers.convert('124')=='en to fire')
        self.assert_(numbers.convert('12.4')==numbers.errormsg)
        self.assert_(numbers.convert('jensen')==numbers.errormsg)
        self.failUnless(numbers.convert('-13')==numbers.errormsg,'negativ')

def makeTestSuite():
    suite=unittest.TestSuite()
    suite.addTest(testconversion('testConvert'))
    return suite

suite = makeTestSuite()
unittest.TextTestRunner(verbosity=2).run(suite)
```

Vi starter med følgende forsøk på å skrive selve modulen:

```
"""
Converting a natural number as string to numberwords
s=convert(n)
"""
errormsg='ikke et tall'

def convert(n):
    return errormsg
```

Hvis vi kjører (run) testmodulen får vi følgende resultat:

```
testConvert (__main__.testconversion) ... FAIL

======================================================================
FAIL: testConvert (__main__.testconversion)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "F:\html\ml\pytest\testnumbers.py", line 10, in testConvert
    self.assert_(numbers0.convert('124')=='en to fire')
AssertionError

----------------------------------------------------------------------
Ran 1 test in 0.016s

FAILED (failures=1)
```

Vi skriver om modulen vår:

```
# -*- coding: cp1252 -*-
"""
Converting a natural number as string to numberwords
s=convert(n)
"""
errormsg='ikke et tall'
wds=['null','en','to','tre','fire','fem','seks','sju','åtte','ni']

def convert(n):
    if n.isdigit():
        res=''
        for c in n:
            res+=wds[int(c)]+' '
        return res[:-1]
    else:
        return errormsg
```

Vi tester igjen og får følgende testresultat:

```
testConvert (__main__.testconversion) ... ok

----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
```

Eksempel 2

Utgangspunktet er følgende "kravspesifikasjon":

Vi ønsker å skrive en modul som leser en fil fra filsystemet eller fra nettet, fjerner alle tagger med innhold og lager en oversikt over hvor mange ganger hver bokstav, a..å, forekommer i den rensede teksten. Vi skiller ikke mellom store (versaler) og små (minuskler) bokstaver. Resultatet skal rapporteres som en dictionary med bokstaven som nøkkel og antall forekomster som verdi.

Det første jeg gjør er å legge en plan for hvordan jeg ønsker å lage implementasjonen. Jeg kan skrive dette som en sammenhengende kodeblokk i modulen, eller jeg kan dele det opp i funksjoner. Jeg velger det siste, kalle modulen charstat, og lager følgende skjelett:

```
# -*- coding: cp1252 -*-
"""
loading from anywhere, remove tags and
count letters, produce result as a dictionary
"""
import urllib

errormsg='feil'

#----------------------------------------
# load data from anywhere
def loadData(address):
    return (errormsg)


#----------------------------------------
# clean taggedcontent
def cleanData(T):
    return '<>'

#----------------------------------------
# count characters and store in a dictionary
def countData(T):
    return {}

def makeStatistics(address):
    t=loadData(address)
    if t!=errormsg:
        t=cleanData(t)
        resDict=countData(t)
```

Vi kunne planlagt etter mange andre strukturer, f.eks. en klasse eller en samlet metode eller rett og slett flat kode i modulen. Jeg har valgt den inndelingen jeg har for å kontrollere stegene i funksjonaliteten bedre. Jeg kommer til å teste koden deretter.

Det neste jeg gjør er å lage en modul som skal teste koden min.

```
"""
testing charstat
"""
#import the module we want to test
import charstat
#import the tester
import unittest

class testcharcounter(unittest.TestCase):

    def setUp(self):
        # set up a list of different address  we want to handle
        self.addresses=["c:\\articles\\ml\\pytest\\wordfile1.txt",
                        "wordfile1.txt",
                        "http://www.ia.hiof.no/~borres/ml/index.shtml",
                        "http://www.ia.hiof.no/~borres/ml/python/frej1.txt",
                        "wordfile1.txt"]

    def testLoadData(self):
        # test loading
        for ad in self.addresses:
            self.failUnless(charstat.loadData(ad)!=charstat.errormsg)

    def testClean(self):
        # test cleaning
        for ad in self.addresses:
            T=charstat.loadData(ad)
            T=charstat.cleanData(T)
            self.failUnless(T.find('<')==-1)
            self.failUnless(T.find('>')==-1)


    def testCount(self):
        # test counting
        T=charstat.loadData(self.addresses[0])
        T=charstat.cleanData(T)
        count=charstat.countData(T)
        self.failUnless(count['e']==5)

def makeTestSuite():
    suite=unittest.TestSuite()
    # comment/uncomment as we go
    suite.addTest(testcharcounter('testLoadData'))
    suite.addTest(testcharcounter('testClean'))
    suite.addTest(testcharcounter('testCount'))
    return suite

suite = makeTestSuite()
unittest.TextTestRunner(verbosity=2).run(suite)
```

Hvis vi kjører (run) denne modulen får vi følgende resultat:

```
testLoadData (__main__.testcharcounter) ... FAIL

======================================================================
FAIL: testLoadData (__main__.testcharcounter)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "F:\html\ml\pytest\testcharstat.py", line 22, in testLoadData
    self.failUnless(charstat0.loadData(ad)!=charstat0.errormsg)
AssertionError

----------------------------------------------------------------------
Ran 1 test in 0.016s

FAILED (failures=1)
```

det gikk dårlig. Dette er ikke overraskende siden vi ikke har implementert loadData. Vi vender tilbake til modulen vår og forsøker oss med en implementasjon. Etter litt prøving og feiling sitter vi kanskje med følgende:

```
# -*- coding: cp1252 -*-
"""
loading from anywhere, remove tags and
count letters, produce a dictionary
"""
import urllib

errormsg='feil'

#----------------------------------------
# load data from anywhere
def loadData(address):
    try:
        # as an url, possibly with scheme file:
        f=urllib.urlopen(address)
        res=f.read()
        f.close()
        return (res)
    except:
         # try as a strait filename (absolute or relative)
         try:
            file=open(address,'r')
            res=file.read()
            file.close()
            return (res)
         except:
            return (errormsg)


#----------------------------------------
# clean taggedcontent
def cleanData(T):
    tag_is_on=False
    res=''
    for ch in T:
        if ch =='<':
            tag_is_on=True
        elif ch =='>':
            tag_is_on=False
        elif not tag_is_on:
            res=res+ch
    return res

#----------------------------------------
# count characters and store in a dictionary
def countData(T):
    countables='abcdefghijklmnopqrstuvwxyzæøå'
    res={}
    for ch in countables:
        res[ch]=0
    T=T.lower()
    for ch in T:
        if countables.find(ch)!= -1:
            res[ch]+=1
    return res

def makeStatistics(address):
    t=loadData(address)
    if t!=errormsg:
        t=cleanData(t)
        resDict=countData(t)
```

Vi tester igjen, og resultatet av testen blir:

```
testLoadData (__main__.testcharcounter) ... ok
testClean (__main__.testcharcounter) ... ok
testCount (__main__.testcharcounter) ... ok

----------------------------------------------------------------------
Ran 3 tests in 0.563s

OK
```

Merk at vi kan velge hvilke tester vi ønsker å kjøre ved å bestemme hvordan vi bygger opp testsuite:

```
def makeTestSuite():
    suite=unittest.TestSuite()
    suite.addTest(testcharcounter('testLoadData'))
    suite.addTest(testcharcounter('testClean'))
    suite.addTest(testcharcounter('testCount'))
    suite.addTest(testcharcounter('testGetStatistics'))
    return suite
```

Det kan stilles mange spørsmål til de testene som er laget.

- Som det framgår har jeg testet flere filer som jeg når på forskjellig måte. Dersom en fil ikke eksisterer skal programmet kunne handtere dette. Jeg har ikke eksplisitt testet dette.
- Jeg har testet tellingen mot et kjent resultat i en fil: self.assert_(dict['e'] == 5). Det kan diskuteres om dette er bra nok.
- Videre kan en diskutere detaljer i alle testfunksjonene og testene kan lett lage mer omfattende.

Det kan være en bra øvelse og bygge ut testbatteriet. Kopier filene charstat.py og testcharstat.py og forsøk deg fram.

Referanser og eksempler:

De aktuelle kodebitene kan kopieres direkte fra siden: charstat.py og charstattest.py.

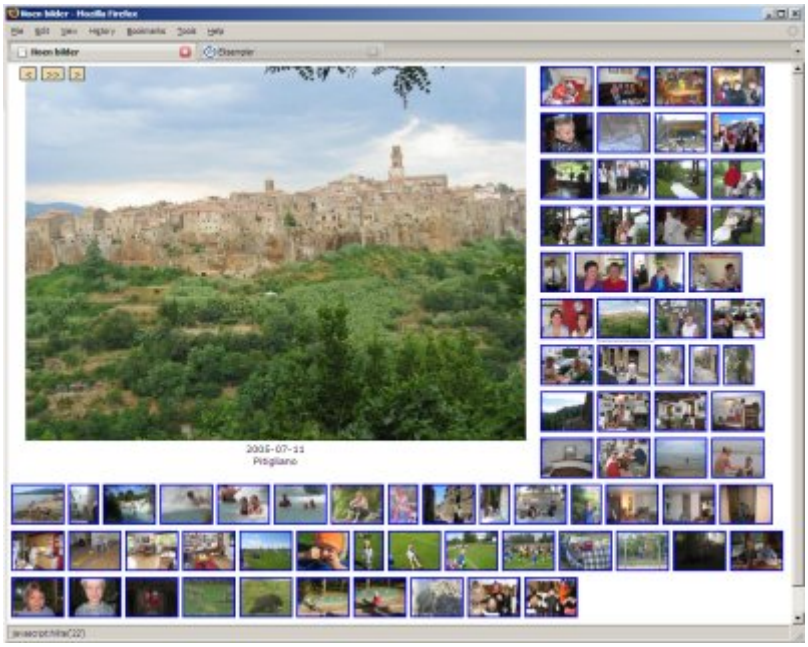
- Pythondokumetasjonen: www.python.org/doc/
- PyUnit på SourceForge.net: pyunit.sourceforge.net
- Boka "Dive into Python" ligger på nett og har et kapittel om PyUnit: www.diveintopython.org/toc/index.html

Bildegalleri

En enkel ordning av bilder for framvisning

Hva:

Et enkelt bildegalleri



Utgangspunktet er at vi har en katalog med bilder av varierende størrelse. Vi antar for enkelhets skyld at det er JPG-bilder. Vi skal lage en halvautomatikk som lar oss vise et utdrag av bildene i et enkelt galleri, med thumbnails.

Vi ønsker følgende funksjonalitet:

- Vi skal se en full versjon av den thumbnailen vi klikker på
- Vi skal kunne vandre framover og bakover
- Vi skal kunne ha et automatisk "slideshow"

Vi skal lage løsningen på følgende måte:

1. Vi lager en XML-fil som inneholder en kopling mellom bildenavn, dato når bildet er tatt(hvis det er et fotografi) og en kort beskrivelse, **data.xml**
2. Vi lager en master for en framvisningsside, **master.html**
3. Vi lager en Pythonmodul, **makegalleri.py**, som leser data.xml, master.html og produserer **index.html** som er den siden som skal vise fram galleriet.
4. Vi skriver en javascript-fil, **galleri.js**, som inkluderes i index.html og tar seg av dynamikken på framvisningssiden,
5. Vi skriver css-fil, **galleri.css**, som justerer layouten for oss.

Datafil

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<bilder>
    <front>
    <![CDATA[
       <h1>Noen bilder</h1>
       <p>Her finner du  litt diverse bilder
       </p>
    ]]>
    </front>
    <bilde>
        <imagename>bilde1</imagename>
        <dato>2007-03-30</dato>
        <text>Pussig fyr</text>
    </bilde>
        ...
</bilder>
```

Master for indexfil

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en" xml:lang="en">

<head>
    <link href="galleri.css" rel="STYLESHEET" />
    <title>Noen bilder</title>
    <script type="text/javascript" src="galleri.js"> </script>
</head>

<body onload="init()">

  <!-- navigator -->
  <div  class="printhide" style="position:absolute;left:17px;top:7px">
      <span class="mybutton" onclick="previous()">&lt;</span>
      <span class="mybutton" id="autobutton" onclick="toggle()">&gt;&gt;</span>
      <span class="mybutton"  onclick="next()">&gt;</span>
  </div>

<div style="float:left">
  <div <!-- STYLE FROM PYCODE --> id="hilitedimg">

  </div>
</div>

<div id="content" class="printhide">
            <!-- CONTENT FROM PYCODE -->
</div>

<div style="clear:left;text-align:center">
    <hr/>
    <p>
        En enkel demonstrasjon.
        Skaler vinduet så ser du at
        de små bildene flyter.
        Avspillingskontrollen burde kanskje flyttes ?
    </p>
    <p>
        <a href="javascript:history.back()">Tilbake</a>
    </p>
</div>

</body>
</html>
```

Vi har satt inn to kommentarer som skal erstattes med materiale som produseres i Pythonmodulen.

Pythonmodul

Merk at modulen bruker Python Imaging Library (PIL) til å skalere ned bildet til en akseptabel størrelse og til å lage thumbnails.

```
import xml.dom.minidom
import Image
import os
"""
See: http://www.ia.hiof.no/~borres/ml/galleri/p-galleri.html
"""



thumbFragment="""
<span>
<a href="javascript:hilite('%s')" title="Vis dette">
<img alt="?" src="images/%s" />
</a>
<div style="display:none" id="thumb%s">
  <div><img src="images/%s.jpg" alt="-"/></div>
  <div ><!-- datoen -->
     %s
  </div>
  <div><!-- teksten -->
    %s
  </div>
</div>
</span>
"""

#------------------------
# Read / write text files
def getTextFile(filename):
    try:
        file=open(filename,'r')
        res=file.read()
        file.close()
        return res
    except:
        print 'Trouble reading: '+filename
        return None

def storeTextFile(filename,txt):
    try:
        file=open(filename,'w')
        file.write(txt)
        file.close()
    except:
        print 'Trouble writing to: '+filename

#-------------------------------------------
# extract text from a node
def getText(nodelist):
    rc = ''
    for node in nodelist:
        if node.nodeType == node.TEXT_NODE or \
           node.nodeType == node.CDATA_SECTION_NODE:
            t=node.data.encode('ISO-8859-1')
            rc += t
    return rc

#-------------------------------------------
# produce a thumbnail
def makeThumb(infile,outfile):
    global maxWidth
    global maxHeight
    im = Image.open(infile)
    w=im.size[0]
```

```
    if w > maxWidth:
        maxWidth=w
    h=im.size[1]
    if h > maxHeight:
        maxHeight=h
    tsize=T_SIZE,T_SIZE
    if w > h:
        tsize= int(round(1.0*w/h*T_SIZE)),int(round(1.0*w/h*T_SIZE))
    im.thumbnail(tsize,Image.ANTIALIAS)
    im.save(outfile,"JPEG")

#------------------------------------------------
# resize an image, down to max height and width
def resizeImage(imfile,maxW,maxH):
    im = Image.open(imfile)
    w=im.size[0]
    h=im.size[1]
    if w <= maxW and h <= maxH:
        return
    if w > maxW:
        h=int(round((1.0*h/w)*maxW))
        w=maxW
    if h > maxH:
        w=int(round((1.0*w/h)*maxH))
        h=maxH
    isize=w,h
    im=im.resize(isize,Image.ANTIALIAS)
    im.save(imfile,"JPEG")


#------------------------------------------------
# build DOM tree from XML-file
def buildDom(filename):
    T=getTextFile(filename)
    if T== None:
        return None
    try:
        return xml.dom.minidom.parseString(T)
    except:
        print 'Error building dom '
        return None

#------------------------------------------------
# Find all image elements and populate
# a spanFragment list
def buildPage(dom,catalog):
    global masterPage
    res=''
    # pick up frontpage
    front=''
    try:
        front=getText(dom.getElementsByTagName('front')[0].childNodes)
    except:
        front='<p>Bilder</p>'
    res="""
    <div style="display:none" id="thumb0">
    %s
    </div>
    """%front

    # pick up imagedescriptions
    bildeliste=dom.getElementsByTagName('bilde')
    ix=0
    for bilde in bildeliste:
```

```
        ix+=1
        # pick up imagename (106_0629), dato(2006-04-14)
        # and txt(dette er en tekst)
        try:
            imx=bilde.getElementsByTagName('imagename')[0]
            imagename=getText(imx.childNodes)
            if imagename.endswith('.jpg'):
                imagename=imagename.replace('.jpg','')
            dato=getText(bilde.getElementsByTagName('dato')[0].childNodes)
            txt=getText(bilde.getElementsByTagName('text')[0].childNodes)
            imagefile=catalog+'images\\'+imagename+'.jpg'
            thumbfile=catalog+'images\\t_'+imagename+'.jpg'
            resizeImage(imagefile,I_WIDTH,I_HEIGTH)
            makeThumb(imagefile,thumbfile)
            res+='\n'+thumbFragment%\
            (str(ix),'t_'+imagename+'.jpg',str(ix),imagename,dato,txt)
        except:
            print 'something went wrong,bilde nr: '+str(ix)
    style='style="text-align:center;width:%spx;height:%spx"'%\
          (str(maxWidth+5),str(maxHeight+40))
    masterPage=masterPage.replace('<!-- STYLE FROM PYCODE -->',style)
    masterPage=masterPage.replace('<!-- CONTENT FROM PYCODE -->',res)
    return masterPage

#-------------
catalog='f:\\html\\ml\\galleri\\'
#-------------------------------------------
# max size for images to show
I_WIDTH=700
I_HEIGTH=500
# will be set as max image dimensions found
maxWidth=20
maxHeight=20
# max heigth for thumbnail
T_SIZE=50

masterPage=getTextFile(catalog+'master.html')
if masterPage==None:
    exit(0)
dom=buildDom(catalog+'data.xml')
if dom != None:
    res=buildPage(dom,catalog)
    try:
        outfile=open(catalog+'index.html','w')
        outfile.write(res)
        outfile.close()
    except:
        print 'Error writing file ',filename
```

Javascript

```
var currentNr=0;
var maxNr=1;
var stopped=true;

function init()
{
  var nr=0;
  while (document.getElementById('thumb'+nr))
   nr+=1
  maxNr=nr-1;
  hilite('0');
}

function hilite(nr)
{
  var content=document.getElementById('thumb'+nr).innerHTML;
  document.getElementById('hilitedimg').innerHTML=
    '<div>'+content+'</div>';
  currentNr=parseInt(nr);
}

function previous()
{
    if(currentNr==0)
      hilite(maxNr);
    else
      hilite(currentNr-1);
}

function next()
{
  if(currentNr==maxNr)
    hilite(0);
  else
    hilite(currentNr+1);
}

function toggle()
{
  if(stopped)
  {
    stopped=false;
    setTimeout("show()",3000);
    document.getElementById('autobutton').innerHTML=
      ' || ';
  }
  else
  {
    stopped=true;
    document.getElementById('autobutton').innerHTML=
      ' &gt;&gt; ';
  }
}

function show()
{
  if (!stopped)
  {
    next();
    setTimeout("show()",3000);
  }
}
```

Stilsett

```
@media screen
    {
        body{
                color: black;
                font-family:"Verdana","Geneva","Arial","Times",sans-serif;
                font-size:13px;
                margin:5px;
                padding:0px;
        }
        .text{font-size:14px}
        .dato{color:gray}
        .mybutton{cursor:pointer;color:black;
                  background-color:#EED9B0;
                  border-style:solid;
                  border-width:thin;
                  padding:5px;
                  font-weight:bold;
                  margin-right:2px}
        h2{background-color:#EED9B0;}
    }

    @media print
    {
        .printhide{display:none;font-size:16px;text-align:center}
    }
```

DEMO: http://www.ia.hiof.no/~borres/ml/galleri/index.html

Referanser og eksempler:

Python Imaging Library, PIL: www.pythonware.com/products/pil/

Et komplett eksempel: sample.zip

Slideshow

Hva:

En enkel bildeframviser

Dette eksempelet bygger på noe av det samme resonnementet som i modulen Bildegalleri, men kan leses uavhengig av denne modulen. Vi antar at vi har en katalog med bilder, og vi ønsker å lage en vevside som viser disse bildene i tilfeldig rekkefølge.

Vi skal lage:

- En vevside, **auto.html**
- Et Javascript, **auto.js**
- Et Pythonscript som går på tjeneren, **auto.py**
- Et stilsett, **auto.css**

Vevsiden

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en" xml:lang="en">

<head>
    <title>Bilde sekvens</title>
    <link href="auto.css" rel="STYLESHEET" />
    <script type="text/javascript" src="auto.js"> </script>
</head>

<body onload="init()">
<h2 class="printhide">Slideshow</h2>
<div id="hilitedimg" style="height:560px;text-align:center">
    <p>
        <img src="wait.gif" alt="Starter straks"/>
    </p>
</div>

<div class="printhide">
  <p id="toggle" onclick="toggleShow()" class="mybutton">
      Start
  </p>
  <hr/>
  <p>
        <a href="javascript:history.back()">Tilbake</a>
    </p>
</div>

</body>
</html>
```

Javascriptet

Løsningen er basert på at vi laster inn en randomisert liste av bilder. Dette gjøres i JSON-format når siden lastes, funksjonen **init**. Hvordan JSON-formatet brukes går fram av funksjonen: **processJSON** nedenfor.

Denne løsningen innebærer at den tilfeldige rekkefølgen gjentar seg. Det er lett å modifisere Javascriptkoden slik at vi får en ny tilfeldig liste når den første er "oppbrukt".

Funksjonene **toggleShow** og **Show** tar seg av selve bildevisningen. De resterende funksjonene realiserer AJAX.

```
var imList=null;
var currentNr=0;
var stopped=true;
var delay=4000;
var scriptURL='<scriptadressen>';
var imageCatalog='<absolutt adresse på tjeneren>';
var imageURL='<absolutt url ti bildekatalogen>';

function establishRequest()
{
    var theRequest=null;
    if (window.XMLHttpRequest)
        theRequest = new XMLHttpRequest();
    else if (window.ActiveXObject)
    {
        try { theRequest = new ActiveXObject("Msxml2.XMLHTTP");
        }
        catch(e) {
            try {
              theRequest= new ActiveXObject("Microsoft.XMLHTTP");
            }
            catch(e) {
              theRequest=null;
            }
        }
    }
    else
        theRequest=null;
    return theRequest;
}

function postRequest(theRequest,params,theURL)
{
    theRequest.onreadystatechange =
        function( ) {
                    processJSON(theRequest);
        };
    theRequest.open("POST", theURL,true);
    theRequest.setRequestHeader("Content-type",
          "application/x-www-form-urlencoded");
    theRequest.setRequestHeader("Content-length", params.length);
    theRequest.setRequestHeader("Connection", "close");
    theRequest.send(params);
}

function processJSON( aRequest)
{
    if (aRequest.readyState == 4) {
        if ((aRequest.status == 200) || (aRequest.status == 304))
        {
            try{
              p = eval("(" +aRequest.responseText+")");
              imList=p.imList;
              toggleShow();
            }
            catch(ex){
              document.getElementById('hilitedimg').innerHTML=
               'Finner ingen bilder';
            }
        }
        else
            alert("Problems accessing data:\n" + aRequest.statusText);

    }
```

```
}

function doRequest(theURL,params)
{
    var myRequest=establishRequest();
    if (myRequest)
    {
        postRequest(myRequest,params,theURL);
    }
    else
      {
        alert('Nettleseren nekter å hente data');
    }
}

  function toggleShow()
  {
       if(stopped)
        {
          stopped=false;
          show()
          document.getElementById('toggle').innerHTML='Stopp';
        }
        else
        {
          stopped=true;
          document.getElementById('toggle').innerHTML='Start';
        }
  }

  function show()
  {
    if (!stopped)
    {
      currentNr+=1;
      if (currentNr >= imList.length)
          currentNr=0
      imsrc=imageURL+imList[currentNr].adr;
      imalt=imList[currentNr].adr;
      imtag='<img src="'+imsrc+'" alt='+imalt+'"/>'
      document.getElementById('hilitedimg').innerHTML=imtag;
      setTimeout("show()",delay);
    }
  }

function init()
{
    params='catalog='+imageCatalog;
    doRequest(scriptURL,params)
}
```

Pythonscriptet

```
#! /usr/bin/python2
"""
Scans a directory and returns a randomized list of images
use see: http://www.ia.hiof.no/~borres/ml/slideshow/p-slideshow.html
"""
import os,random
import cgi
import cgitb; cgitb.enable()
import operator


def buildImageList(catalog):
    imageList=[]
    for root, dirs, files in os.walk(catalog, topdown=False):
        for name in files:
            # only jpg-files
            if not name.endswith('.jpg'):
                continue
            # drop thumbs
            if name.startswith('t_'):
                continue
            imageList.append(name)
    if len(imageList)==0:
        return '{"imList":[{"adr":"Finner ingen bilder i katalogen"}]}'
    random.shuffle(imageList)
    T=''
    for im in imageList:
        T+='{"adr":"%s"},'%im
    return '{"imList":[%s]}'%T[:-1]


form=cgi.FieldStorage()
print 'Content-type: text/plain\n'

#expect to find: catalog

if form.has_key('catalog'):
    catalog=form['catalog'].value
    print buildImageList(catalog)
else:
    print '{"imList":[{"adr":"Finner ingen bilder"}]}'
```

Stilsettet

Stilsettet inneholder mulighet for å skrive ut bildene i ren form, uten menyer og overskrifter.

```
@media screen
{
    body{
            color: black;
            font-family:"Verdana","Geneva","Arial","Times",sans-serif;
            font-size:13px;
            margin:5px;
            text-align:center;
            padding:0px;
    }
    .text{font-size:14px}
    .dato{color:gray}
```

```
    .mybutton{cursor:pointer;color:black;
              background-color:#EED9B0;
              border-style:solid;
              border-width:thin;
              margin-right:2px}
    h2{background-color:#EED9B0;}
}
@media print
{
    .printhide{display:none;font-size:16px;text-align:center}
}
```

DEMO: http://www.ia.hiof.no/~borres/ml/slideshow/auto.html

Referanser og eksempler:

- Modulen AJAX
- Modulen JSON

Temperaturkurver

Hva:

Temperaturkurver

Vi skal bruke PIL til å lage kurver som sammenligner temperaturer. Løsningen er svært enkel og lar seg lett utvide.

Vi henter data fra vevsiden Rimfrost, se referanser nederst. En fil, temp-oslo.txt, med data for et sted (Oslo) kan typisk se slik ut:

temperaturer for Oslo

Hver linje inneholder årstallet, 12 månedsgjennomsnitt og årsgjennomsnittet. alle verdiene er tabulator separert.

Løsningen er AJAX-basert. det vil si at vi bestemmer byer og år, sender dette til et pythonscript og får en img-tag tilbake. Det er et problem med dette. navnet på bildet med kurver på må endres ved hver forespørsel for at vi skal få automatisk oppdatering av vevsiden. Dette er ved at serverskriptet hele tiden lager et nytt filnavn, se Pythonscriptet nedenfor.

Vi skal altså lage:

- en HTML-side, **showtemp.html**, som holder løsningen.
- et Javascript, **showtemp.js**, som tar seg av AJAX-kommunikasjonen med tjeneren.
- et Pythonscript, **tempil.html**, som lager bildet med kurver på og returnerer en image-tag.

HTML-siden

DEMO: http://www.ia.hiof.no/~borres/ml/temperaturer/showtemp.html

Javascriptet

Vi benytter oss av AJAX-biblioteket som er beskrevet i modulen Kodestrategier. Vi trenger derfor bare en funksjon som pakker ned parametere:

```
function showCurve(form)
{
    var by1=form.by1.value;
    var by2=form.by2.value;
    var year=form.year.value;
    var params='by1='+by1+'&by2='+by2+'&year='+year;
    doRequest('http://www.ia.hiof.no/~borres/cgi-bin/ajax/temperaturer/tempil.py',
              params,
              'curve');
}
```

Pythonscript

```
#! /usr/bin/python2
"""
    Simple construction of temperature curves for two cities.
    New image is constructed and a img-tag is returned.
    name of file is randomized to ensure reload in webpage
    See: http://www.ia.hiof.no/~borres/temperaturer/p-temperaturer.html
    Note that default font and size is used.
    Limitations on fonts on Linux without config
"""
import cgi,os,random
import Image,ImageDraw,ImageFont,ImageFilter
import cgitb; cgitb.enable()


#-------------------------
# global constants
# canvas size:
canW=500
canH=400
# degree resolution
deltaY=4
deltaX=40
# curve colors
ccolors=[(255,0,0),(0,255,0),(0,0,255)]
# will save with a different name to force reload of webpage
newname='f'+str(random.randint(1,10000))+'.jpg'

#------------------------
# remove all .jpg files
def removeJPGs():
    list=os.listdir('.')
    for f in list:
        if f.endswith('.jpg') and f.find('canvas')==-1:
            os.remove(f)

#--------------------------
# Establish temperatur dictionary from a file
# Key is year, value is list of 12 month temperaturs
def makeTemp(filename):
    T=None
    try:
        file=open(filename,'r')
        T=file.read()
        file.close()
    except:
        return None
    lines=T.split('\n')
    temps={}
    for line in lines:
        line=line.strip()
        if line.startswith('//'):
            continue
        parts=line.split('\t')
        if len(parts) != 14:
            continue
        temps[parts[0].strip()]=parts[1:-1]
    return temps

#--------------------------
# make missing message in canvas
def makeError(canvas,year):
    draw=ImageDraw.Draw(canvas)
    draw.text((100,200),
              "Finner ikke data for "+year,
```

```
                fill=(0,0,0))
    canvas.save(newname, "JPEG")
    del draw

#-----------------------------------------
# set up a std canvas with grids etc
def makeCanvas():
    # months
    months=['X','J','F','M','A','M','J','J','A','S','O','N','D',]
    # set up a canvas
    canvas=Image.new("RGB",(canW,canH),(255,255,255))
    draw=ImageDraw.Draw(canvas)
    # draw grid
    # horizontals
    for ix in range(50,-50,-10):
        draw.line((0,canH/2-ix*deltaY)+(canW-deltaX/2,canH/2-ix*deltaY),
                  fill=(128,128,128))
        draw.text((5,canH/2-ix*deltaY), str(ix),
                  fill=(0,0,0))
    # verticals
    for ix in range(1,13):
        draw.line((ix*deltaX,canH-15)+(ix*deltaX,0),
                  fill=(128,128,128))
        draw.text((ix*deltaX-5,canH-16), months[ix],
                  fill=(0,0,0))
    # horizontal 0-axes
    draw.line((0,canH/2)+(canW,canH/2),fill=(255,0,0))
    del draw
    return canvas

#---------------------------
# make curve for a certain year
def makeCurve(canvas,temps,year,title,curveix):
    # try to pick up data
    if not temps.has_key(year):
        makeError(canvas,year)
        return

    # make curve data with degrees
    strvals=temps[year]
    vals=[]
    for v in strvals:
        tmp=int(round(float(v),0))*deltaY
        vals.append( ( (len(vals)+1)*deltaX,canH/2-tmp ) )

    draw=ImageDraw.Draw(canvas)
    # draw the curve
    draw.line(vals, fill=ccolors[curveix],width=2)

    # draw year
    draw.text((deltaX,curveix*16),title,fill=ccolors[curveix])
    del draw
    return canvas


#-------------------------------
# clean up, remove old resultfile(s)
removeJPGs()

form=cgi.FieldStorage()
print 'Content-type: text/plain\n'

#expect to find: by1,by2,year
by1='oslo'
```

```
by2='moskva'
year='2000'
if form.has_key('by1'):
    by1=form['by1'].value
if form.has_key('by2'):
    by2=form['by2'].value
if form.has_key('year'):
    year=form['year'].value
fil1='temp-%s.txt'%by1
fil2='temp-%s.txt'%by2
temps1=makeTemp(fil1)
temps2=makeTemp(fil2)
canvas=makeCanvas()
canvas=makeCurve(canvas,temps1,year,by1+'-'+year,0)
canvas=makeCurve(canvas,temps2,year,by2+'-'+year,1)

canvas.save(newname, "JPEG")
print """<img src="http://www.ia.hiof.no/~borres/cgi-bin/ajax/temperaturer/%s"
             alt="canvas"/>"""%newname
```

Utvidelser

Det er ganske lett å lage utvidelser til denne funksjonaliteten. Et nærliggene forslag er å lage søylediagrammer istedet for kurver. Videre er det klart at det er enkelt og greitt å ta inn flere byer og flere år.

Referanser og eksempler:

Python Imaging Library, PIL: www.pythonware.com/products/pil/
Data fra Rimfrost: www.rimfrost.no/
Modulen: AJAX

Land i verden

Hva:

Land og hovedsteder i verden

Denne modulen demonstrerer alternative datastrukturer i Python. Vi bruker to datasett som beskriver alle land i verden, en CSV-fil og en XML-fil.

Datagrunnlaget er hentet fra www.geonames.org. Datasettene som er brukt er kopiert lokat og er beskrevet i modulen: Noen datasett.

Datasettene kan kopieres herfra: CSV-fila: land.txt og XML-fila: land.xml

Vi skal gjøre følgende øvelser på disse dataene:

- Lese CSV-fila og rapportere land fordelt på verdensdel
- Lese XML-fila og rapportere land fordelt på verdensdel
- Lese XML-fila parse med SAX og rapportere land fordelt på verdensdel
- Lese CSV-fila og opprette en liste av land-objekter og rapportere derfra
- Lese XML-fila og opprette en liste av land-objekter og rapportere derfra
- Lage et fleksibelt Land objekt

Vi skal altså ikke gjøre noe særlig "nyttig". Hensikten er bare å illustrerer noen Python mekanismer.

Lese CVS

Vi leser inn CVS-fila, og rapporterer land fordelt på verdensdel.

-

```
"""
Read land.txt
Report countries sorted on continent
"""
#------------------------
# helper to read a file
def getTextFile(filename):
    try:
        file=open(filename,'r')
        res=file.read()
        file.close()
        return res
    except:
        print 'Trouble reading: '+filename
        return None
#-------------------------------
# continets [code,fullname,collected countries]
continents=[['EU','Europa',[]],
            ['AS','Asia',[]],
            ['NA', 'North America',[]],
            ['AN','Antarctica',[]],
            ['SA','South america',[]],
            ['OC','Oceania',[]],
            ['','No region',[]]]

#-------------------------------
# wherever you place the file:
T=getTextFile('..\\..\\commondata\\geonames\\land.txt')
if T!=None:
    lines=T.split('\n')
    for line in lines:
        parts=line.split('\t')
        if len(parts)!=12:
            continue
        (iso2,iso3,isonum,fips,country,capital,
         area,population,continent,language,money,gid)=parts
        for c in continents:
            if c[0]==continent:
                c[2].append( country)
    #--------------------
    # report, collected in a string, and ready for filewrite
    result=''
    for c in continents:
        result+= c[1]+'\n'
        for co in sorted(c[2]):
            result+='\t'+co+'\n'
    print result
```

Lese XML

Vi leser inn XML-fila, og rapporterer land fordelt på verdensdel.

-

```
import xml.dom.minidom
"""
Read land.xml
Report countries sorted on continent
"""
#------------------------
# helper to read a file
def getTextFile(filename):
    try:
        file=open(filename,'r')
        res=file.read()
        file.close()
        return res
    except:
        print 'Trouble reading: '+filename
        return None
#-----------------------
# helper to collect all text in a node
def getText(nodelist):
    rc = ''
    for node in nodelist:
        if node.nodeType == node.TEXT_NODE:
            t=node.data.encode('ISO-8859-1')
            rc += t
    return rc

#-------------------------------
# continets [code,fullname,collected countries]
continents=[['EU','Europa',[]],
            ['AS','Asia',[]],
            ['NA', 'North America',[]],
            ['AN','Antarctica',[]],
            ['SA','South america',[]],
            ['OC','Oceania',[]],
            ['','No region',[]]]

#-------------------------------
# wherever you place the file:
T=getTextFile('..\\..\\commondata\\geonames\\land.xml')
if T!=None:
    dom=xml.dom.minidom.parseString(T)
    countries=dom.getElementsByTagName('country')
    for co in countries:
        country=getText(co.getElementsByTagName('countryName')[0].childNodes)
        continent=getText(co.getElementsByTagName('continent')[0].childNodes)
        for c in continents:
            if c[0]==continent:
                c[2].append( country)
    #--------------------
    # report, collected in a string, and ready for filewrite
    result=''
    for c in continents:
        result+= c[1]+'\n'
        for co in sorted(c[2]):
            result+='\t'+co+'\n'
    print result
```

Lese og parse XML med SAX

Vi leser inn XML-fila, og rapporterer land fordelt på verdensdel.

-

```
import xml.parsers.expat
"""
Read land.xml
Parse with SAX
Report countries sorted on continent
"""
#------------------------
# helper to read a file
def getTextFile(filename):
    try:
        file=open(filename,'r')
        res=file.read()
        file.close()
        return res
    except:
        print 'Trouble reading: '+filename
        return None

#------------------------------
# continets [code,fullname,collected countries]
continents=[['EU','Europa',[]],
            ['AS','Asia',[]],
            ['NA', 'North America',[]],
            ['AN','Antarctica',[]],
            ['SA','South america',[]],
            ['OC','Oceania',[]],
            ['','No region',[]]]

#-----------------------------------------------------
# 2 flags to remember type of element
# 2 globals to store wanted data
# 3 primitive handler functions used by saxparser
country=''
continent=''
countryFlag=False
continentFlag=False
#---------------------
# opening an element
def start_element(name, attrs):
    global countryFlag
    global continentFlag
    if name=='countryName':
        countryFlag=True
    elif name=='continent':
        continentFlag=True
#---------------------
# closing an element
def end_element(name):
    global countryFlag
    global continentFlag
    global country
    global continent
    global continents
    countryFlag=False
    continentFlag=False
    if name=='country':
        for c in continents:
            if c[0]==continent:
                c[2].append( country)
#---------------------
# content of an element
def char_data(data):
    global country
    global continent
```

```
    global continents
    if continentFlag:
        continent=data
    elif countryFlag:
        country=data

#--------------------------------
# wherever you place the file:
p = xml.parsers.expat.ParserCreate()
p.returns_unicode = 0
# tell parser about our callbacks
p.StartElementHandler = start_element
p.EndElementHandler = end_element
p.CharacterDataHandler = char_data

T=getTextFile('..\\..\\commondata\\geonames\\land.xml')
if T!=None:
    p.Parse(T)
    #--------------------
    # report, collected in a string, and ready for filewrite
    result=''
    for c in continents:
        result+= c[1]+'\n'
        for co in sorted(c[2]):
            result+='\t'+co+'\n'
    print result
```

Lese CSV til objekter

Vi leser inn csv-fila, ordner i objektliste og rapporterer land fordelt på verdensdel.

-

```
"""
Read land.txt
build list of objects
Report countries sorted on continent
"""
#------------------------
# defining a class
class Country:
    def __init__(self, name,continent):
        self.name=name
        self.continent=continent

#------------------------
# helper to read a file
def getTextFile(filename):
    try:
        file=open(filename,'r')
        res=file.read()
        file.close()
        return res
    except:
        print 'Trouble reading: '+filename
        return None
#-------------------------------
# continets [code,fullname,collected countries]
continents=[['EU','Europa',[]],
            ['AS','Asia',[]],
            ['NA', 'North America',[]],
            ['AN','Antarctica',[]],
            ['SA','South america',[]],
            ['OC','Oceania',[]],
            ['','No region',[]]]

#--------------------------------
# wherever you place the file:
T=getTextFile('..\\..\\commondata\\geonames\\land.txt')
allCountries=[]
if T!=None:
    lines=T.split('\n')
    for line in lines:
        parts=line.split('\t')
        if len(parts)!=12:
            continue
        allCountries.append(Country(parts[4],parts[8]))
    #--------------------
    # report, collected in a string, and ready for filewrite
    result=''
    for c in continents:
        result+= c[1]+'\n'
        for co in allCountries:
            if co.continent==c[0]:
                result+='\t'+co.name+'\n'
    print result
```

CSV eller XML til objekter

Skisse av en løsning som skal kunne handtere begge formatene , ordne i objektliste og rapporterer land fordelt på verdensdel.

-

```
import xml.dom.minidom
"""
Read land.txt or land.xml
build list of objects
Report countries sorted on continent
"""
#------------------------
# defining a class
class Country:
    def __init__(self, clist=None,cnode=None):
        if clist!=None:
            self.name=clist[4]
            self.continent=clist[8]
        elif cnode!=None:
            cnn=cnode.getElementsByTagName('countryName')[0]
            self.name=getText(cnn.childNodes)
            ccc=cnode.getElementsByTagName('continent')[0]
            self.continent=getText(ccc.childNodes)
        else:
            self.name='langvekkistan'
            self.continent=''


#------------------------
# helper to read a file
def getTextFile(filename):
    try:
        file=open(filename,'r')
        res=file.read()
        file.close()
        return res
    except:
        print 'Trouble reading: '+filename
        return None
#-----------------------
# helper to collect all text in a node
def getText(nodelist):
    rc = ''
    for node in nodelist:
        if node.nodeType == node.TEXT_NODE:
            t=node.data.encode('ISO-8859-1')
            rc += t
    return rc
#-------------------------------
# continets [code,fullname,collected countries]
continents=[['EU','Europa',[]],
            ['AS','Asia',[]],
            ['NA', 'North America',[]],
            ['AN','Antarctica',[]],
            ['SA','South america',[]],
            ['OC','Oceania',[]],
            ['','No region',[]]]


#--------------------------------
allCountries=[]
#------ this section-----
#T=getTextFile('..\\..\\commondata\\geonames\\land.txt')
#if T!=None:
#    lines=T.split('\n')
#    for line in lines:
#        parts=line.split('\t')
#        if len(parts)!=12:
#            continue
```

```
#        allCountries.append(Country(clist=parts))

#------ or this section-----
T=getTextFile('..\\..\\commondata\\geonames\\land.xml')
if T!=None:
    dom=xml.dom.minidom.parseString(T)
    countries=dom.getElementsByTagName('country')
    for co in countries:
        allCountries.append(Country(cnode=co))
#--------------------
# report, collected in a string, and ready for filewrite
result=''
for c in continents:
    result+= c[1]+'\n'
    for co in allCountries:
        if co.continent==c[0]:
            result+='\t'+co.name+'\n'
print result
```

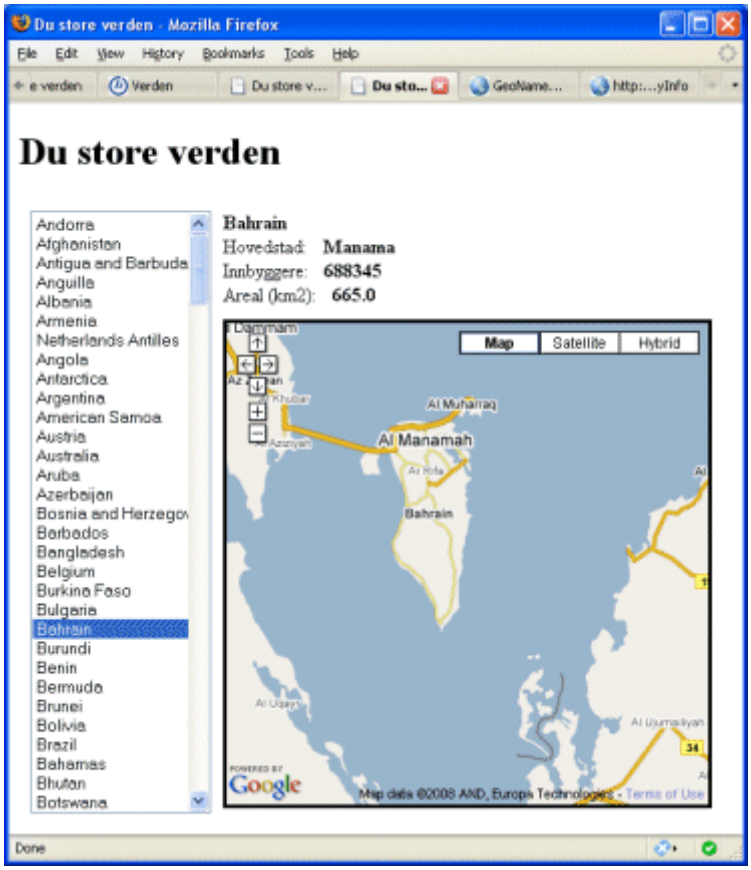
Referanser og eksempler:

GeoNames www.geonames.org/

Land i verden

Hva:

Kart (Google Maps) over alle land i verden



I denne modulen skal vi lage en enkel vevside som lar oss finne fram noe informasjon om land i verden. Utgangspunktet er data fra geonames [1] . De XML-dataene vi skal bruke er referert på modulen Noen datasett, og har adresse: http://www.ia.hiof.no/~borres/commondata/geonames/land.xml. XML-fila ser slik ut: land.xml

Vi bruker GoogleMaps [2] . Merk at for å bruke Google Maps må du skaffe deg en ID som gjelder for et domene.

Dette lar seg løse på mange måter. Vi kan lage en helt enkel vevside som oppdateres ettersom vi gjør valg, eller vi kan lage en side som er AJAX-basert, dvs. den henter bare de dataene vi trenger. Vi skal imidlertid gjøre det ennå enklere: Vi lager en HTML-side som er selvforsynt. Det vi si at vi skal kunne velge nye land uten å hente data fra tjeneren.

DEMO: http://www.ia.hiof.no/~borres/cgi-bin/gmaps/verden.py

Det er to komponenter som skal lages:

- Et Pythonscript som henter nødvendige data og apkker dem pent sammen på en HTML-side, som så leveres til klienten
- Et javascript som tar seg av dialogen med brukeren, og som altså er selvforsynt med de data som er pakket på vevsiden

python scriptet

Scriptet er ganske enkelt. Det henter XML-fila, bygger et DOM-tre og lager en listeboks. Denne plasseres inn på et vevsideskjelett som så skrives tilbake. Listeboksen bygges opp slik at den inneholder alle de data vi trenger for å tegne nytt kart. Noen linjer i boksen:

```
<option value="Nassau,301790,13940.0,-78.978897,26.929165,-72.737503,20.915276,Bahamas">
Bahamas
</option>
<option value="Thimphu,2232291,47000.0,88.751938,28.325275,92.115265,26.703049,Bhutan">
Bhutan
</option>
```

Dataene som ligger som value er i tur og orden: capital,population,areaInSqKm,bBoxWest,bBoxNorth,bBoxEast,bBoxSouth,countryName. Dette tilsvarer elementer til hvert country-element i vår kildefil, land.xml. Box-dataene er hjørner i det rektangelet som omgir landet.

Pythonscriptete er i sin helhet slik:

Javascriptet

Javascriptet leser de dataene som er pakket ned i listeboksen, plasserer dem i dertil angitte elementer, og bruker spesielt box-datene til å sette opp data for å tegne kart.

Referanser og eksempler:

- [1] Geonames:
  geografiske data
  www.geonames.org/
- [2] Google Maps:
  Map API
  code.google.com/apis/maps/index.html

Koden er sitert i teksten

Beregning av avstand i tid

Bruk av pyUnit til utvikling

Hva:

Avstand i tid, en webtjeneste

Jeg vil lage en webtjeneste som returnerer hvor mange dager det er mellom to datoer. Løsningen skal være enkel og den skal nåes via en URL + parametere. Det eneste som skal returneres fra tjenesten er et tall, antall dager. Eller dersom feil : ?. Jeg vil ha tre former for beregninger:

- **until**, typisk bruk:<url>?until&date=2007-12-24
- **since**, typisk bruk:<url>?since&date=2006-12-24
- **between**, typisk bruk:<url>?between&date=2007-12-24&date2=2008-12-24

Jeg vil videre ha det slik at dersom året angis som xxxx, skal det bety nærmeste forekomst av datoen, enten dette året, neste år(until) eller forrige år(since). xxx for år i between skal tolkes slik at date betraktes som since og date2 betraktes som until.

Videre vil jeg at dersom det gis inn gale eller mangende formater, ikke eksisterende datoer eller andre feil, skal tjenesten returnere: ?. Tjenesten skal altså tåle hva som helst og aldri gi annen output enn et tall, positivt eller negativt, eller ?.

Jeg vil forsøke å løse dette ved å "lage testen før jeg skriver koden", ved hjelp av pyUnit. Utviklingen er beskrevet nedenfor i noen steg.

DEMO: http://www.ia.hiof.no/~borres/ml/days/daytester.html

Steg1

Vi begynner med å lage en kodestrategi. Jeg vil gjøre det slik at jeg lager en funksjon for until-beregninger, en for since-beregninger og en for between-beregninger. Disse vil jeg kalle på grunnlag av de parametrne modulen mottar. Tolking av disse parametrene vil jeg teste senere. Nå konsentrerer jeg meg om de tre funksjonene. Skjelettet for modulen min blir slik:

```
"""
    This module will calculate the difference in days
    between today and any other date, or between two dates
"""

import datetime

def daysUntil(thisDay,datstr):
    """ thisDay is reference date, datstr is target date in string format"""
    return '?'

def daysSince(thisDay,datstr):
    """ thisDay is reference date, dat is target date in string format"""
    return '?'

def daysBetween(dat1str,dat2str):
    """ the two involved dates in string format"""
    return '?'
```

Så etablerer jeg testen slik:

```
"""
Testing the module daydiff
"""
# import the module we want to test:
import daydiff
# import library for making dates
import datetime
# import the PyUnit class we want to subclass:
import unittest

class testdatediff(unittest.TestCase):

    def setUp(self):
        # set up a refrence date that will be used
        # in stead of datetime.date.today()
        self.refdate=datetime.date(2007,9,15)

    def testUntil(self):
        """ testing the until-case """
        for d in [['2007-09-15','0'],
                  ['2007-09-16','1'],
                  ['2007-08-15','-31'],
                  ['xxxx-09-25','10'],
                  ['xxxx-09-05','356'],
                  ['09-05','?'],
                  ['2006:09-05','?']]:
            print d
            self.assert_(daydiff.daysUntil(self.refdate,d[0])==d[1])


    def testSince(self):
        """ testing the since-case """
        for d in [['2007-09-15','0'],
                  ['2007-09-16','-1'],
                  ['xxxx-09-16','364'],
                  ['xxxx-09-14','1']
                  ]:
            print d
            self.assert_(daydiff.daysSince(self.refdate,d[0])==d[1])

    def testBetween(self):
        """ testing the between-case """
        for d in [['2007-09-16','2007-09-17','1'],
                  ['2007-09-16','2007-09-11','-5'],
                  ['2007-09-16','xxxx-09-17','1'],
                  ['2007-09-16','xxxx-09-15','365'],
                  ['xxxx-09-15','xxxx-09-14','365']
                  ]:
            print d
            self.assert_(daydiff.daysBetween(d[0],d[1])==d[2])

def makeTestSuite():
    suite=unittest.TestSuite()
    suite.addTest(testdatediff('testUntil'))
    suite.addTest(testdatediff('testSince'))
    suite.addTest(testdatediff('testBetween'))
    return suite

suite = makeTestSuite()
unittest.TextTestRunner(verbosity=2).run(suite)
```

Første forsøk på å teste går naturlig nok galt, alle de første testsettene for hver funksjon feiler:

```
testing the until-case ... ['2007-09-15', '0']
FAIL
testing the since-case ... ['2007-09-15', '0']
FAIL
testing the between-case ... ['2007-09-16', '2007-09-17', '1']
FAIL
.......
```

Steg2

Jeg bearbeider funksjonene og får noe slikt.

```
"""
    This module will calculate the difference in days
    between today and any other date, or between two dates
"""

import datetime

def makeLegalDateParts(datstr):
    """ make legal y,m,d from the string datstr. y=xxxx is legal"""
    prts=datstr.split('-')
    if len(prts) != 3:
        return [-1,-1,-1]
    (y,m,d)=prts
    m=int(m.lstrip('0'))
    d=int(d.lstrip('0'))
    if y.isdigit():
        y=y.lstrip('0')
        return [y,m,d]
    if y=='xxxx':
        return [y,m,d]
    return [-1,-1,-1]

def daysUntil(thisDay,datstr):
    """ thisDay is reference date,
        datstr is target date in string format"""
    (y,m,d)=makeLegalDateParts(datstr)
    if y == -1:
        return '?'
    if y.isdigit():
        try:
            dat=datetime.date(int(y),m,d)
            return str((dat-thisDay).days)
        except:
            return '?'
    # we have an xxxx situation
    for y in range(thisDay.year,thisDay.year+2):
        try:
            dat=datetime.date(y,m,d)
            if dat > thisDay:
                return str((dat-thisDay).days)
        except:
            return '?'
    return '?'

def daysSince(thisDay,datstr):
    """ thisDay is reference date,
        dat is target date in string format"""
    (y,m,d)=makeLegalDateParts(datstr)
    if y == -1:
        return '?'
    if y.isdigit():
        try:
            dat=datetime.date(int(y),m,d)
            return str((thisDay-dat).days)
        except:
            return '?'
    # we have an xxxx situation
    for y in range(thisDay.year,thisDay.year-2,-1):
        try:
            dat=datetime.date(y,m,d)
            if dat < thisDay:
                return str((thisDay-dat).days)
        except:
            return '?'
```

```
    return '?'

def daysBetween(dat1str,dat2str):
    """ the two involved dates in string format"""
    (y1,m1,d1)=makeLegalDateParts(dat1str)
    (y2,m2,d2)=makeLegalDateParts(dat2str)
    if y1==-1 or y2==-1:
        return '?'
    if y1.isdigit():
        try:
            return daysUntil(datetime.date(int(y1),m1,d1),dat2str)
        except:
            return '?'
    elif y2.isdigit():
        try:
            return daysFrom(datetime.date(int(y1),m1,d1),dat2str)
        except:
            return '?'
    else:
        # we have two xxxx cases, we make a choice
        try:
            return daysUntil(datetime.date(datetime.date.today().year,m1,d1),
                             dat2str)
        except:
            return '?'
    return '?'
```

Testkoden er som før og jeg får:

```
testing the until-case ... ['2007-09-15', '0']
['2007-09-16', '1']
['2007-08-15', '-31']
['xxxx-09-25', '10']
['xxxx-09-05', '356']
['09-05', '?']
['2006:09-05', '?']
ok
testing the since-case ... ['2007-09-15', '0']
['2007-09-16', '-1']
['xxxx-09-16', '364']
['xxxx-09-14', '1']
ok
testing the between-case ... ['2007-09-16', '2007-09-17', '1']
['2007-09-16', '2007-09-11', '-5']
['2007-09-16', 'xxxx-09-17', '1']
['2007-09-16', 'xxxx-09-15', '365']
['xxxx-09-15', 'xxxx-09-14', '365']
ok

----------------------------------------------------------------------
Ran 3 tests in 0.063s

OK
```

Vi slår oss til ro med at vi har skrevet tre testede funksjoner.

Steg3

Nå skal vi bruke de tre funksjonene i en modul som ligger på nettet og besvarer spørsmål. Vi har nå flere veier å gå når vi skal fortsette og teste. Jeg velger å lage meg en vevside som tester modulen. Igjen skriver jeg testen før jeg går videre med koden.

DEMO: http://www.ia.hiof.no/~borres/ml/days/_full_test.html

Det som testes er nå den ferdige modulen:

```
#!/usr/bin/python2
"""
    This module will calculate and return the difference in days
    between today and any other date, or between two dates.
    See: http://www.ia.hiof.no/~borres/cgi-bin/ajax/days/explained.html
"""
import datetime
import os

def makeLegalDateParts(datstr):
    """ make legal y,m,d from the string datstr. y=xxxx is legal"""
    prts=datstr.split('-')
    if len(prts) != 3:
        return [-1,-1,-1]
    (y,m,d)=prts
    m=int(m.lstrip('0'))
    d=int(d.lstrip('0'))
    if y.isdigit():
        y=y.lstrip('0')
        return [y,m,d]
    if y=='xxxx':
        return [y,m,d]
    return [-1,-1,-1]

def daysUntil(thisDay,datstr):
    """ thisDay is reference date, datstr is target date in string format"""
    (y,m,d)=makeLegalDateParts(datstr)
    if y == -1:
        return '?'
    if y.isdigit():
        try:
            dat=datetime.date(int(y),m,d)
            return str((dat-thisDay).days)
        except:
            return '?'
    # we have an xxxx situation
    for y in range(thisDay.year,thisDay.year+2):
        try:
            dat=datetime.date(y,m,d)
            if dat > thisDay:
                return str((dat-thisDay).days)
        except:
            return '?'
    return '?'

def daysSince(thisDay,datstr):
    """ thisDay is reference date, dat is target date in string format"""
    (y,m,d)=makeLegalDateParts(datstr)
    if y == -1:
        return '?'
    if y.isdigit():
        try:
            dat=datetime.date(int(y),m,d)
            return str((thisDay-dat).days)
        except:
            return '?'
    # we have an xxxx situation
    for y in range(thisDay.year,thisDay.year-2,-1):
        try:
            dat=datetime.date(y,m,d)
            if dat < thisDay:
                return str((thisDay-dat).days)
        except:
            return '?'
```

```
    return '?'

def daysBetween(dat1str,dat2str):
    """ the two involved dates in string format"""
    (y1,m1,d1)=makeLegalDateParts(dat1str)
    (y2,m2,d2)=makeLegalDateParts(dat2str)
    if y1==-1 or y2==-1:
        return '?'
    if y1.isdigit():
        try:
            return daysUntil(datetime.date(int(y1),m1,d1),dat2str)
        except:
            return '?'
    elif y2.isdigit():
        try:
            return daysFrom(datetime.date(int(y1),m1,d1),dat2str)
        except:
            return '?'
    else:
        # we have two xxxx cases, we make a choice
        try:
            return daysUntil(datetime.date(datetime.date.today().year,m1,d1),
                             dat2str)
        except:
            return '?'
    return '?'

if __name__=="__main__":

    import cgi
    form=cgi.FieldStorage()
    #---------------------------
    # expect to find parameters:
    # diff= until | since | between
    # diff=until | since -> date
    # dif=between -> date and date2
    #---------------------------
    thisDay=datetime.date.today()
    # if requested from testpage we set reference date
    if os.environ.has_key('HTTP_REFERER')\
       and str(os.environ['HTTP_REFERER']).find('_full_test.html') != -1:
        thisDay=datetime.date(2007,9,15)

    print 'Content-type: text/plain\n'
    diff=''
    date1=''
    date2=''
    if form.has_key('diff'):
        diff=form['diff'].value
    if form.has_key('date'):
        date1=form['date'].value
    if form.has_key('date2'):
        date2=form['date2'].value

    if diff=='until':
        print daysUntil(thisDay,date1)
    elif diff=='since':
        print daysSince(thisDay,date1)
    elif diff=='between':
        print daysBetween(date1,date2)
    else:
        print '?'
```

Til slutt

har vi en modul daydiff.py, vi har en testmodul, daytest.py og vi har en testside, fulltest,html. Hele testbatteriet er intakt og vi kan test senere når vi oppdager feil, eller dersom vi ønsker å skrive om koden. (det forekommer meg at den kan forenkles noe ?)

Den ferdige modulen kan brukes herfra: http://www.ia.hiof.no/~borres/cgi-bin/ajax/days/daydiff.py.

Det er laget en forklaringsside: http://www.ia.hiof.no/~borres/cgi-bin/ajax/days/explained.html.

Referanser og eksempler:

Se modulen: Pythontesting