INF3331 Oblig2, runde 2

Gard Inge Rosvold

November 4, 2014

Svar på første tilbakemelding

NOTE: Dette er en kopi av REVIEW2_NO som heter REVIEW2_NO_INPUT. Har kun kopiert/endret for å få linjene korte nok til å passe på siden. Dette fordi jeg ikke tok meg tid til å ta i bruk listings-pakka istedetfor FancyVerbatim. Teksten er selvfølgelig uendra.

```
Er rapporten velskrevet, og beskriver den de viktigste delene av
oppgaveløsningen?
Rapporten mangler / det er noe feil med den.

Er implementasjonen verifisert/testet?
For uten at jeg kan kjøre programmet, så finnes det ingen andre
former for tester. doctest, pytest eller nosetest finnes ikke.
Compile.py fungerte ikke på min maskin.

Er det lett å forstå hvordan programmet kjøres?
Det er svært enkelt ettersom argparse er implementert på en god måte.

Løser programmene de problemene de skal?
Oppgave 12 mangler, 7 fikk jeg ikke til å kjøre. 1-4, 6, 8, 10, 11
kjører. Oppgave 13 finnes heller ikke, fordi rapporten mangler.

Er programmene lette å lese og forstå?
God kompilering og bruk av verbose gjør det enkelt å forstå gangen
i programmet, samt hvordan det er bygget opp.
```

Det var tilbakemeldinga. Oppgave 12 eksisterte ikke, det stemmer. Fordi jeg da hadde ingen funksjoner som faktisk kunne testes. Og skreiv derfor dette i rapporten. Har nå laga en test_suite.py for å prøve å teste. Men de failer. Forstår ikke helt hvorfor, da jeg ikke ser "ulikheten" i de mellom 'left == right'. Bedre forklaring i riktig seksjon av rapporten. Oppgave 13 derimot eksisterte, jeg hadde bare klart å sende med feil fil. Oblig2.pdf som lå på github den dagen var oppgaveteksten (tror jeg) - men hadde sendt med Oblig2.tex og Oblig2_after.tex som var riktige. Hadde personene brukt mine test-filer ville de sett rapporten og kanskje godkjent likevel. Helt til slutt var det en minibug i kompilatoren når den leste ut navnet, som tok 2 min å fikse. Dette kunne tester gjort og forklart

hva som var galt (og samtidig funnet ut at kompilatoren fungerte helt fint). Det var det, oppgaven skal være "så godt som perfekt" nå.

Introduksjon

Generelle funksjoner for 'prepro.py'

```
__name__ == "__main__"
```

Main funksjonen kjører og håndterer in/utfiler. Den sjekker argumentene som ble sendt med til programmet, og setter variable ift. disse (f.eks verbose). Den aktiverer også fancy verbatim hvis den er sendt med. Til slutt aktiverer den funksjonen handle_lines.

```
handle_lines(in_file, out_file, in_line, fancy, verbose):
    """
Handle lines of un-processed file, and executes
correct writing to pre-processed file

Parameters
----------
in_file : file
    Current file being read(preprocessed)

out_file : file
    The file to write pre-processes to

in_line : string
    current line being processed

fancy : boolean
    Use fancy verbatim

verbose : boolean
    Be loud about what to do
    """
```

Denne funksjonens rolle er å lese og skrive riktig kode til den nye utfilen. Om det er samme linje som gamle eller ny ift. bestemte parametre, bestemmer funksjonen. Ved de fleste ekstra funksjonaliteter implementert i obligen kaller den funksjonen 'handle_command'.

```
handle_command(in_file, in_line, fancy, verbose):
    """
Handle pre-processor command from un-processed file

Parameters
----------
in_file : file
    The latex document currently being pre-processed

in_line : string
    The current line being worked on in latex document
```

```
fancy : boolean
    Use fancy verbatim

verbose : boolean
    Be loud about what to do

Returns
-------
The content to write : string
    """
```

Siste generelle funksjonen håndterer hva som skal skje ift. den nye funksjonaliteten implementert i obligen, og kaller på korrekt funksjon med korrekt parametre.

*Oppgave 1 - Importering av skript

Funksjoner

```
import_script(in_file, in_line, fancy, verbose):
    """
Handles the command '%@import [filename "regex"]',
which imports code from current or 'filename' document
according to regex-expression.
Or '%@exec' which prints out execution-style of following code.

Parameters
----------
in_file : file
    The latex document currently being read from

in_line : string
    The current line being processed

fancy : boolean
    Use fancy verbatim

verbose : boolean
    Be loud about what to do

Returns
-------
script : string
    """
```

Forklaring

Funksjonen håndterer alle tilleggsfunksjonaliteter som starter med enten

- %@import file "regex-expression"
- %@import
- eller %@exec

Ved punkt 1 (%@import file "regex-expression") henter programmet ut kode fra filen nevnt ved hjelp av regex-uttrykket (se denne obligens orginal-kode for eksempel). Ved de to andre leser den bare linjer fram til neste %@ og skriver dette ut i verbatim eller ved Fancy Verbatim ift. om det er kode (%@import) eller eksekvering (%@exec).

*Oppgave 2 - Eksekvering av skript

Funksjoner

```
execute_script(in_file, in_line, fancy, verbose):
    """
Executes the file after %@exec with given parameters

Parameters
----------
latex_file : file
    The latex document currently being pre-processed

in_line : string
    Current line being processed

fancy : boolean
    Use fancy verbatim

verbose : boolean
    Be loud about what to do

Returns
-------
executed script : string
"""
```

Forklaring

Ved tilleggsfunksjonalitetskommando '%@exec language file arg(s)' kjører denne funksjonen den nevnte fila med angitte argumenter, og printer ut en passende latex-terminal utskrift av kjøring og resultat.

Eksempel

```
\%@exec test\_script.py -2
```

Terminal

```
$ python test_script.py -2
2.0
```

Oppgave 3+4 - Kodeformatering med vanlig innliming av kode

Funksjoner

```
begin_shaded_verbatim(fancy):
    """
Prints correct begin{Verbatim} to file

Parameters
----------
fancy : boolean
    Produces fancy verbatim if true

Returns
-------
Start of verbatim : string
    """
```

```
end_shaded_verbatim(fancy):
    """
Prints correct end{verbatim} to file

Parameters
----------
fancy : boolean
    Produces fancy verbatim if true

Returns
-------
End of Verbatim : string
    """
```

```
begin_execution_verbatim(fancy):
    """
Prints correct begin{Verbatim} to file

Parameters
----------
fancy : boolean
    Produces fancy verbatim if true

Returns
-------
Start of execution environment : string
    """
```

```
end_execution_verbatim(fancy):
    """
Prints correct end{Verbatim} to file

Parameters
----------
fancy : boolean
```

```
    Produces fancy verbatim if true

Returns
-------
End of execution-environment : string

Example
-------
>>> end_execution_verbatim(False)
'\\\\end{verbatim}\\n\\\\noindent\\n'
    """
```

Forklaring

Nødvendige
usepackage{package} blir henta gjennom main hvis argumentet -f eller –fancy er aktivert i kallet på prepro.py. De fire andre funksjonene printer ut korrekt start/slutt på verbatim ift. om fancy er aktivert eller ei. Siden oppgave 4 basicly er copy-paste ved bruk av %@import og %@exec viser jeg eksempel på bruk av disse under.

Eksempel

%@import med tilhørende %@exec eksempel:

```
\%@exec
$ echo "help I'm stuck in a small shell-script"
\%@
```

Terminal

```
$ echo "help I'm stuck in a small shell-script"
```

Oppgave 5 - Kode-eksekvering

Funksjoner

```
execute_code(in_file, in_line, fancy, verbose):
    """
Executes written code from the un-processed latex-file and prints the
terminal-values into the new pre-processed file.

Parameters
----------
in_file : file
    The latex document currently being pre-processed

in_line : string
    Current line being processed
```

```
fancy : boolean
    Use fancy verbatim

verbose : boolean
    Be loud about what to do

Returns
-------
executed code : string
    """
```

Forklaring

Først blir en ny falsk fil oppretta med navnet i tilleggsfunksjonalitetkommandoen. I fila blir koden mellom %@python|bash file_name args og %@ lagt inn. Dette skrives som en blue shader ved Fancy Verbatim for å vise koden i latex-dokumentet. Deretter blir dette kjørt med execute_script funksjonen som forklart i oppgave 2 og vises ved Fancy Verbatim i latex-terminal-vindu.

Eksempel

```
\%@bash fake_name.sh fake_arg
echo "2+2" | bc
\%@
```

```
$ bash fake_name.sh fake_arg
echo "2+2" | bc
```

Terminal

```
$ bash fake_name.sh fake_arg
4
```

```
\%@python fake_name.py fake_arg
print 2+2
\%@
```

```
$ python fake_name.py fake_arg
print 2+2
```

Terminal

```
$ python fake_name.py fake_arg
4
```

*Oppgave 7 - Kompilering av preprosessert LaTeX-fil

Funksjoner

```
preprocess(args, verbose):
    """
Handles a possible preprocess before compiling.

Parameters
----------
args : ArgumentParser
    The arguments parsed

verbose : boolean
    Be loud about what to do
    """
```

```
compile(compilefile, interaction, verbose):
    """
Compiles latex file with given arguments.

Parameters
----------
args : ArgumentParser
    The arguments passed and processed

interaction : string
    Interaction-mode for passing to pdflatex

verbose : boolean
    Be loud about what you do
    """
```

```
printout(out, verbose):
    """
Prints out information about compilation

Parameters
----------
out : string
    The whole output from pdflatex-compilation

verbose : boolean
    Be loud about what to do
    """
```

Forklaring

Main gjør argumentparsing og aktiverer preprocess hvis det er angitt å gjøres først. Videre kompilerer compile funksjonen, og sender videre til printout for en siste utskrift. Oppdelinga av kode er først og fremst for å bedre oversikten.

*Oppgave 8 - Inkludering av filer

Funksjoner

```
include_files(new_filepath, out_file, fancy, verbose):
    """
Un-processed file input/includes other files in need
of pre-processing. This function opens and handles
these new files before continuing with original file.

Parameters
----------
new_filepath : string
    Path to the new file to be included

out_file : file
    The file to write pre-processes to

fancy : boolean
    Use fancy verbatim if true

verbose : boolean
    Be loud about what to do
    """
```

Forklaring

Denne funksjonen iverksettes når handle_lines funksjonen finner en "\input" eller "\include" linje med tilhørende fil-path. Funksjonen åpner den nye fila, og setter i gang en ny handle_lines funksjon i den nye fila, så den både blir lest, skrevet og pre-prossessert med en gang. Når den er ferdig, fortsetter den orginale filen der den slapp. Dette gir også funksjonalitet for å inkludere filer i den inkluderte fila (=rekursivt).

Eksempel

Dette er tex fila 'oppgave8.tex'.

Denne tester kodeeksekveringa fra oppg5:

```
\%@python fake.py
print "Dette var skrevet av python"
\%@
```

```
$ python fake.py
print "Dette var skrevet av python"
```

Terminal

```
$ python fake.py
Dette var skrevet av python
```

*Oppgave 11 - Front-end til preprosessor

Forklaring

Prepro

```
$ python prepro_help.py
import subprocess
proc = subprocess.Popen("python prepro.py -h", shell=True,\
stdout=subprocess.PIPE, stderr=subprocess.STDOUT)
out, err = proc.communicate()
print out
```

Terminal

```
$ python prepro_help.py
usage: prepro.py [-h] [-o OUTFILE] [-f] [-v] infile

Program preprocesses a latex-file ('infile') and produces a new latex-file ('outfile') with
additional functionality

positional arguments:
  infile                Name of the latex-file you want preprocessed

optional arguments:
  -h, --help            show this help message and exit
  -o OUTFILE, --outfile OUTFILE
                        Name of the new file (cannot be equal to infile)
  -f, --fancy_verbatim  produces more fancy verbatim
  -v, --verbosity       increase output verbosity
```

Compile

```
$ python compile_help.py
import subprocess
proc = subprocess.Popen("python compile.py -h", shell=True,\
stdout=subprocess.PIPE, stderr=subprocess.STDOUT)
out, err = proc.communicate()
print out
```

Terminal

```
$ python compile_help.py
usage: compile.py [-h] [-v] [-i] [-p PREPROCESS] compilefile

A latex compiler shortening terminal output to more meaningful printing, and added functionality

positional arguments:
  compilefile           The latex-file to be compiled

optional arguments:
  -h, --help            show this help message and exit
  -v, --verbosity       Be loud about what to do
  -i, --interaction     Sets pdflatex to interaction=nonstopmode to True
```

```
  -p PREPROCESS, --preprocess PREPROCESS
                        Will preprocess using python prepro.py with compilefile as 'outfile'-name,
                        and -f enabled. See prepro.py -h for details
```

---

*Oppgave 12 - Testing og dokumentasjon

Om testing

Jeg er ikke vant til denne typen testing i det hele tatt, så det var helt nytt for meg. Derfor klarte jeg ikke å implementere tester i runde 1, fordi jeg rett og slett ikke hadde laga test-bare funksjoner. Håper det jeg nå har gjort viser at jeg forstår konseptet, så får jeg se om jeg klarer å bruke det mer nevenyttig i framtida.

Pytest

Nå har jeg prøvd å lage en pytest ved navn 'testsuite.py'. Kjøring vises under, der 2 passes og 2 fails. De to som failer forstår jeg ikke, da min egen tilleggsutskrift viser at det egentlig ikke er noe ulikt - men tydeligvis finner pytest noe. Kan dere hjelpe meg å skjønne det?

Terminal

```
$ py.test testsuite.py
====================================== test session starts ======================================
platform linux2 -- Python 2.7.6 -- pytest-2.5.1
collected 4 items

testsuite.py .FF.

============================================ FAILURES ============================================
_____________________________________ test_execute_script _____________________________________

    def test_execute_script():
        in_line = in_file.readline()
        print "in_line: %s" % in_line
        test_content = execute_script(in_file, in_line, fancy, verbose)
        # Egen tilleggsutskrift, skjonner ikke hvorfor disse er ulike..?
        print "Test:\n" + test_content
        print "Expected:\n" + expected_script
>       assert (test_content == expected_script)
E       assert '\\begin{verb...n\\noindent\n' == '\\begin{verba...n\\noindent\n'
E         Skipping 33 identical leading characters in diff, use -v to show
E         - ript.py -2
E         ?         -
E         + ript.py -2
E         - 2.0
E         ?   -
E         + 2.0
E           \end{verbatim}
E           \noindent

testsuite.py:62: AssertionError
```

```
---------------------------------------- Captured stdout -----------------------------------------
in_line: %@exec python test_script.py -2

Test:
\begin{verbatim}
$ python test_script.py -2
2.0
\end{verbatim}
\noindent

Expected:
\begin{verbatim}
$ python test_script.py -2
2.0
\end{verbatim}
\noindent

---------------------------------------- test_execute_code ----------------------------------------

    def test_execute_code():
        in_line = in_file.readline()
        test_content = execute_code(in_file, in_line, fancy, verbose)
        # Egen tilleggsutskrift, skjonner ikke hvorfor disse er ulike..?
        print "Test:\n" + test_content
        print "Expc:\n" + expected_code
>       assert (test_content == expected_code)
E       assert '\\begin{verb...n\\noindent\n' == '\\begin{verba...n\\noindent\n'
E         Skipping 110 identical leading characters in diff, use -v to show
E         - ile.fake 2
E         ?           -
E         + ile.fake 2
E         - 4
E         + 4
E           \end{verbatim}
E           \noindent

testsuite.py:70: AssertionError
---------------------------------------- Captured stdout -----------------------------------------
Test:
\begin{verbatim}
$ python test_file.fake 2
print 2+2
\end{verbatim}
\noindent
\begin{verbatim}
$ python test_file.fake 2
4
\end{verbatim}
\noindent

Expc:
\begin{verbatim}
$ python test_file.fake 2
print 2+2
\end{verbatim}
\noindent
\begin{verbatim}
$ python test_file.fake 2
4
\end{verbatim}
\noindent
```

```
================================== 2 failed, 2 passed in 0.21 seconds ==================================
```

Doc-string test

Ellers har jeg laga en doc-string test for å vise at jeg skjønner konseptet, og hadde jeg skullet klare å lage på de andre metodene ville jeg laga en klasse utav alt så jeg slapp å sende med en "in_file" hele veien :)

*Oppgave 13 - Rapport

Dette er rapporten. Den er skrevet ved hjelp av prepro.py og kompilert med compile.py. Bruk orginalen, .xtex fila eller .pdf fila alt etter personlig smak.