

# Tips för VG-uppgifter Lab 4

---

Jag har fått några mail angående lite strul med VG-uppgifterna. Överlag verkar ni ta er framåt – men jag vill ge er lite tips kring några detaljer som jag tror kan vara lite luriga.

Lägg märke till att vi kan lösa nedanstående problem på andra sätt – så om ni har en lösning som ni tycker känns sund, så behöver ni inte ändra något.

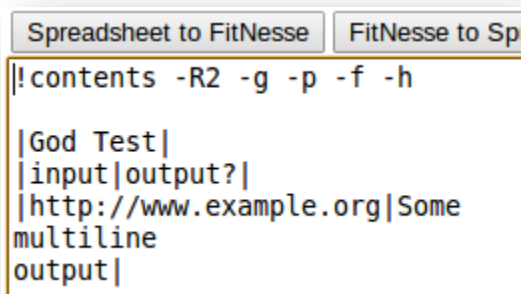
## Literal Text

Wiki-syntaxen kommer i vägen för vissa av VG-uppgifterna i Lab4.

Ett av problemen är att vi behöver hantera input som delvis utgörs av [URL:er](#) (ex: "http://www.example.org/1/"). Denna input kommer tolkas som en faktisk html-länk av wiki-motorn.

Ytterligare ett problem, med samma orsak, som ovanstående problem är att vi behöver hantera output som spänner över flera rader.

En första, naiv, ansats till lösning på ovanstående problem är att vi skriver in följande wiki syntax:

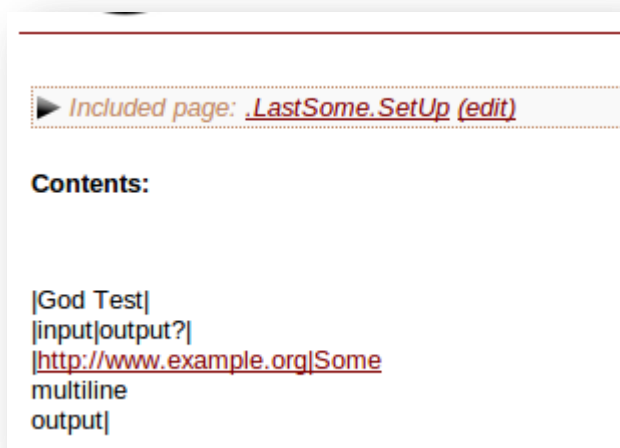


The screenshot shows a text editor window with two tabs: "Spreadsheet to FitNesse" and "FitNesse to Spreadsheet". The active tab is "FitNesse to Spreadsheet". The text in the editor is as follows:

```
!contents -R2 -g -p -f -h

|God Test|
|input|output?|
|http://www.example.org|Some
multiline
output|
```

Ovanstående syntax kommer dock resultera i följande:



The screenshot shows a web browser window displaying the result of the wiki syntax. At the top, there is a message: "Included page: [.LastSome.SetUp \(edit\)](#)". Below this, the word "Contents:" is displayed. The main content area shows the following text:

```
|God Test|
|input|output?|
http://www.example.org|Some
multiline
output|
```

Vi kan se att vår URL tolkas som en html-länk. Vi kan också se att våra radbrytningar gör att syntaxen inte ens tolkas som en tabell längre

Vi vill att texten vi skrivit in i input- och output-kolumnerna inte skall tolkas av wiki-motorn. För att göra detta kan vi använda ett `!-`, `-!` Par – enligt följande bild:

```
|God Test|
|input|output?|
|!-http://www.example.org-!|-Some
multiline
output-!||
```

Den resulterande fitness-tabellen ser då ut som följande:

God Test	
input	output?
http://www.example.org	Some multiline output

Lägg märke till att texten "http://www.example.org" inte längre tolkas som en html-länk och att output-kolumnen innehåller text som spänner över flera rader.

### *Hur löser man ett sånt här problem?*

Vi måste identifiera att texten vi skriver in tolkas annorlunda än vi tänkt – detta syns tydligt – se screenshot ovan. Alternativt så kan vi undersöka hur det vi får in i vår Fixture-klass ser ut genom att t.ex. spara input i en instans-variabel och sedan spotta ut denna instans-variabel i en outputmetod utan någon annan behandling.

Anledningen till att texten tolkas annorlunda är för att FitNesse använder Wiki-syntax och att t.ex. en URL eller en radbrytning har en annan betydelse än en ren textsträng.

När vi väl kommit hit så kan vi dyka ned i FitNesse QuickReference Guide för Wiki-syntax och leta reda på någon syntax som hjälper oss lösa eller kringgå problemet (Alternativt skulle vi kunna googla på problemet och ev. hitta något vettigt).

## Test Double

Flera scenarios i VG-uppgifterna kräver att vi kontrollerar att metoden "puts" anropas på ett output-objekt. Vi hade kunnat implementera en output-klassen själva som vi använder i våra tester (klassen behöver en puts-metod och något sätt att kontrollera vad puts anropats med). Ett alternativ är att vi använder en Test Double av samma slag som användes i Lab 3. Här följer kodlistning:

```
god_test.rb
1 $LOAD_PATH.unshift('../', __FILE__);
2 require 'my_string'
3 require 'rspec/mocks'
4
5 module Fixtures
6   class GodTest
7     def initialize
8       RSpec::Mocks::setup(self)
9     end
10
11     def set_input(arg)
12       @input = arg
13     end
14
15     def output
16       if @input == "http://www.example.org"
17         return "Some\nmultiline\noutput"
18       end
19       @input
20     end
21
22     def set_i_should_see(arg)
23       @i_should_see = arg
24     end
25
26     def pass
27       output = double("output")
28       output.should_receive(:puts).with(@i_should_see)
29       output.puts(@i_should_see)
30       return true
31     end
32   end
33 end
```

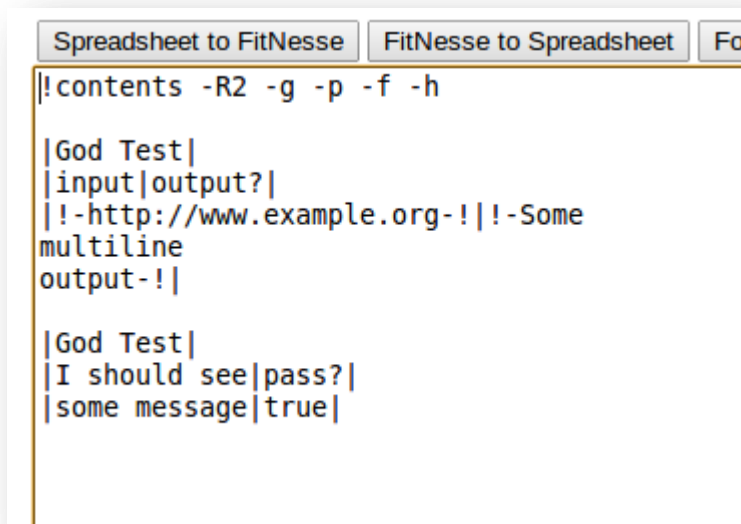
Att lägga märke till:

- double()-metoden hittas i rspec. Vi hade kunnat köra require på "rspec" – detta hade fungerat lika bra. I det här fallet kör jag require på "rspec/mocks" för att jag endast är intresserad av metoderna för att skapa Test Doubles och mocks – inte hela rspec-ramverket.

- För att kunna använda double-metoden utanför rspec-tester behöver jag köra "RSpec::Mocks::setup(self)" i initialize för min fixture-klass.
- Jag returnerar true i min pass-metod. Lägg märke till att should\_receive kommer kasta ett exception ifall puts inte anropas med det förväntade argumentet – dvs. vi kommer inte få ett test som passerar om puts anropas med fel argument. (Testa själva om ni inte tror mig... ;)
- I uppgifterna kommer ni inte anropa .puts direkt på detta sätt – däremot skapar ni ett output-objekt med en should\_receive som ni matar in till Controller-objektet, etc.

## Screenshots

För helhetens skull så bifogar jag screenshots på denna och nästa sida för wiki-syntax och testsida för de färdiga FitNesse-testerna.



```

Spreadsheet to FitNesse  FitNesse to Spreadsheet  For
|!contents -R2 -g -p -f -h
|
|God Test|
|input|output?|
|!-http://www.example.org-!!-Some
multiline
output-!|
|
|God Test|
|I should see|pass?|
|some message|true|

```

✔ **Assertions:** 2 right, 0 wrong, 0 ignored, 0 exceptions (1.745 seconds)

► *Precompiled Libraries*

► *Included page:* [.LastSome.SetUp \(edit\)](#)

**Contents:**

God Test	
input	output?
http://www.example.org	Some multiline output

God Test	
I should see	pass?
some message	true