knitpy: dynamic report generation with python

Jan Schulz

12.03.2015

This is a port of knitr (http://yihui.name/knitr/) and rmarkdown (http://rmarkdown.rstudio.com/) to python.

For a complete description of the code format see http://rmarkdown.rstudio.com/ and replace `{r...}` by `{python ...}` and of course use python code blocks...

Examples

Here are some examples:

```
print("Execute some code chunk and show the result")
```

```
## Execute some code chunk and show the result
```

Codechunks which contain lines without output (e.g. assign the result or comments) will be shown in the same code block:

```
# A comment
text = "All code in the same code block until some output is produced..."
more_text = "...and some more."
print(text)
```

```
## All code in the same code block until some output is produced...
```

```
print(more_text)
```

```
## ...and some more.
```

Code chunk arguments

You can use different arguments in the codechunk declaration. Using `echo=False` will not show the code but only the result.

```
## Only the output will be visible as `echo=False`
```

The next paragraphs explores the code chunk argument `results`.

If 'hide', knitpy will not display the code's results in the final document. If 'hold', knitpy will delay displaying all output pieces until the end of the chunk. If 'asis', knitpy will pass through results without reformatting them (useful if results return raw HTML, etc.)

`results='hold'` is not yet implemented.

```
print("Only the input is displayed, not the output")
```

```
## This is formatted as markdown:
## **This text** will be bold...
```

This text will be bold...

Note: with python code it is recommended to use the IPython/Jupyter display system and an appropriate wrapper (see below) to display such output and not `results="asis"`. This makes it possible to convert such output if the output can't be included in the final format.

You can also not show codeblocks at all, but they will be run (not included codeblock sets `have_run = True`):

```
if have_run == True:
    print("'have_run==True': ran the codeblock before this one.")
```

```
## 'have_run==True': ran the codeblock before this one.
```

Using `eval=False`, one can prevent the evaluation of the codechunk

```
x = 1
```

```
x += 1 # this is not executed as eval is False
```

```
x # still 1
```

```
## 1
```

To remove/hide a codechunk completely, i.e. neither execute it nor show the code, you can use both `eval=False, include=False`: nothing will be shown between this text ...

```
x += 1 # this is not executed and not even shown
```

... and this text here!

The prefix in front of text output (per default `##`) can be changed via the `comment` chunk option to a different string or completely removed by setting it to a empty string `""`or None:

```
print("Text output")
```

```
# result: Text output
```

```
print("Text output")
```

```
Text output
```

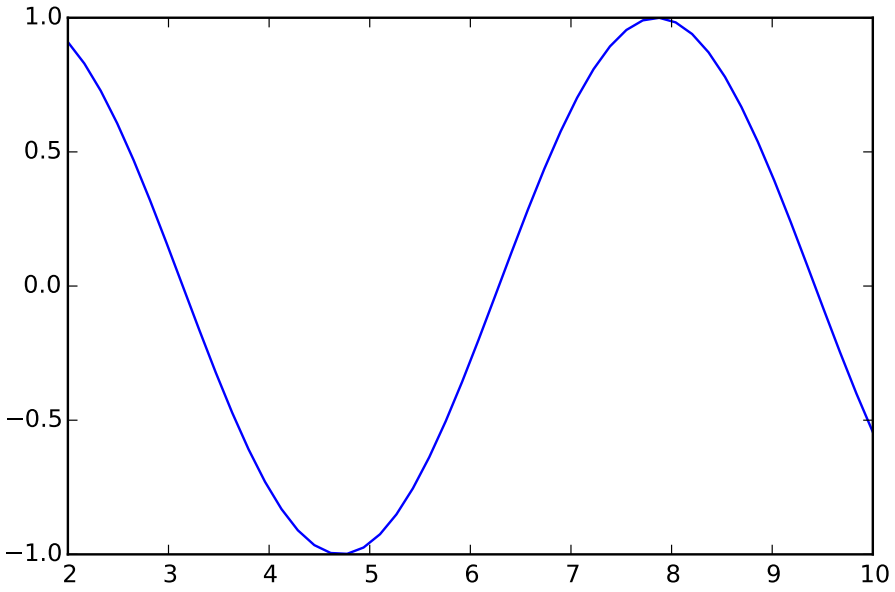
Inline code

You can also include code inline: “m=2” (expected: “m=2”)

IPython / Jupyter display framework

The display framework is also supported.

Plots will be included as images and included in the document. The filename of the plot is derived from the chunk label (“sinus” in this case). The code is not shown in this case (`echo=False`).



If a html or similar thing is displayed via the IPython display framework, it will be included 'as is', meaning that apart from `text/plain`-only output, everything else will be included without marking it up as output. Knitpy automagically tries to include only formats which are understood by pandoc and the final output format (in some case converting the format to one which the final output can handle).

```
from IPython.core.display import display, HTML
display(HTML("<strong>strong text</strong>"))
```

strong text

It even handles `pandas.DataFrames` (be aware that not all formatting can be converted into all output formats):

```
import pandas as pd
pd.set_option("display.width", 200)
s = """This is longer text"""
df = pd.DataFrame({"a":[1,2,3,4,5],"b":[s,"b","c",s,"e"]})
df
```

	a	b
0	1	This is longer text
1	2	b

	a	b
2	3	c
3	4	This is longer text
4	5	e

`pandas.DataFrame` can be represented as `text/plain` or `text/html`, but will default to the html version. To force plain text, use either `print(df)` or set the right `pandas` option:

```
pd.set_option("display.notebook_repr_html", False)
df
```

```
##    a                    b
## 0  1  This is longer text
## 1  2                    b
## 2  3                    c
## 3  4  This is longer text
## 4  5                    e
```

```
# set back the display
pd.set_option("display.notebook_repr_html", True)
```

You can also use package like tabulate together with `results="asis"` or by wrapping it with the appropriate display class:

```
from tabulate import tabulate
from IPython.core.display import Markdown
# either print and use `results="asis"`
print(tabulate(df, list(df.columns), tablefmt="simple"))
```

	a	b
0	1	This is longer text
1	2	b
2	3	c
3	4	This is longer text
4	5	e

```
# or use the IPython display framework to publish markdown
Markdown(tabulate(df, list(df.columns), tablefmt="simple"))
```

	a	b
0	1	This is longer text
1	2	b
2	3	c
3	4	This is longer text
4	5	e

Note that the second version (wrapping it in `Markdown`) is preferred, as this marks the output with the right mimetype and therefore can be converted—if that's needed—to something which the output format understands!

Unfortunately, html tables have to be tweaked for the final output format as e.g. too width tables spill over the page margin in PDF.

Error handling

Errors in code are shown with a bold error text:

```
import sys
print(sys.not_available)
```

ERROR: AttributeError: 'module' object has no attribute 'not_available'

```
AttributeError                            Traceback (most recent call last)
<ipython-input-37-a5971246c0f7> in <module>()
----> 1 print(sys.not_available)

AttributeError: 'module' object has no attribute 'not_available'
```

```
for x in []:
print("No indention...")
```

ERROR: Code invalid