My Project Work Package

Document Number	Equipment or Sub-System
	Work package demonstrator

Subject

How to use the converter to create a work package

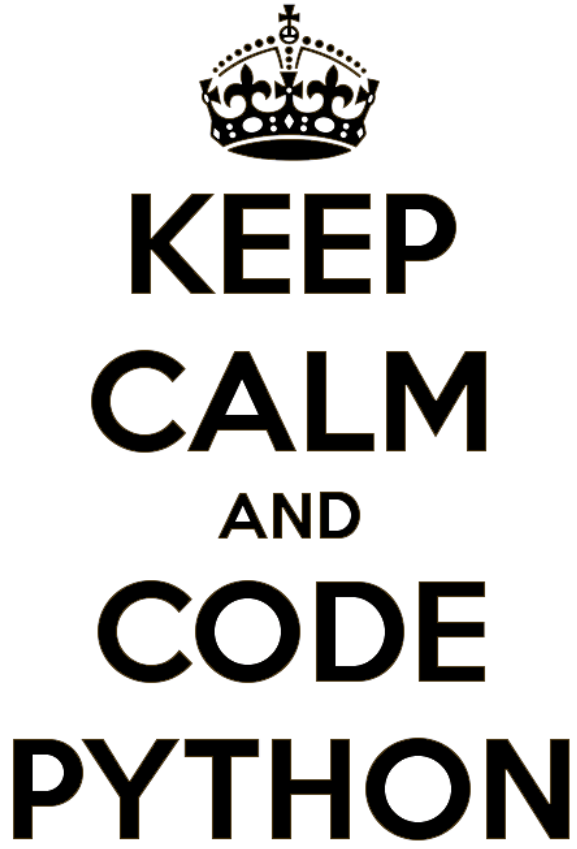
Distribution

Conclusions/Decisions/Amendments

1. This work package demonstrates the use of the converter.
2. It was tested with and should work on Python 3.5 and up.

Author	Signature
CJ Willers	

Date	Date	Date
Previous Package No.	Superseding Package No.	August 25, 2021
		Current Package No.
		0001



ipnb2tex

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Listings

1 The ipnb2tex.py Script

The `ipnb2tex.py` reads the IPython notebook and converts it to a LaTeX set of files (a `*.tex` file and a number of images). The script is invoked as follows:

```
python ipnb2tex.py file.ipynb file.tex imagedir -i -u
```

where

- `file.ipynb` [optional] is the name of the input IPython notebook file. If no input filename is supplied, all `.ipynb` files in current directory will be processed. In this event the output filenames will be the same as the `.ipynb` files, just with a `tex` filetype. The notebook filename must not have spaces.
- `file.tex` [optional] is the name of output LaTeX file. If none is given the output filename will be the same as the input file, but with the `.tex` extension.
- `imagedir` [optional] is the directory where images are written to. If not given, this image directory will be the `./pic` directory.
- `-i` [optional], the lower case letter `i`, if this option is given the code listings are printed inline with the body text where they occur, otherwise listings are floated to the end of the document.
- `-u` [optional] add `\url{}` to the bibtex entries, to obtain `url = {\url{http://ipython.org/}}`, otherwise use the form

  `url = {http://ipython.org/}`

The options must follow after the filenames and dirname and the end of the command line string.

1.1 Known deficiencies

1. Some complex cell-merged HTML tables may not render correctly in LaTeX (let me know if you have such a table).
2. The following HTML elements are not currently processed, these elements are simply ignored: `div`, `img`.
3. The `iframe` HTML element is tested for embedded PDF files which are then included as an image.
4. Coloured text in HTML is not exported as coloured text in LaTeX.
5. Many reserved LaTeX symbols such as hash, caret, underscore and dollar are 'legal' in normal markdown. When rendering to LaTeX these symbols cause errors unless escaped with backslash. In many cases these symbols are escaped, but not always because of context. If the symbols are escaped, they render incorrectly in normal Markdown. Therefore, choose your target renderer and enter the symbols accordingly, accepting problems in the alternative renderer.

2 Heading 1 nb2pdf

Heading 1 is a LaTeX chapter.

Headings down to level 5 (LaTeX paragraph) are supported. Headings at level 6 are treated as normal body text. If you don't need a chapter (i.e., for an article), just don't use a heading at level 1.

When any level of section heading is formed, a label is automatically created from the text in the heading, removing all non-alpha characters. If the section heading is very long the the label will also be long, so keep it short. Also, don't use the same text twice as a heading.

2.1 Heading 2 Second-level heading

2.1.1 Heading 3

2.1.1.1 Heading 4

2.1.1.1.1 Heading 5

Heading 6 some more markdown.

3 Heading X

3.1 Heading X2

3.1.1 Heading X3

3.1.1.1 Heading X4

3.1.1.1.1 Heading X5

Heading X6

3.2 LaTeX Template Format and the Header File

3.2.1 Default Style (Report)

The notebook can be converted to the LaTeX `Report` format with no additional files or document class. A 'standard' notebook, where the first cell is not `Raw NBConvert` will be prepended with a short section of LaTeX code for a standard LaTeX Report. A number of pacakges are also `\usepackage{}`ed, as required by the converted LaTeX code.

3.2.2 User-Defined Style

The notebook can optionally be converted to a user-defined format with minimal effort. The example files included the GitHub distribution provides a 'work package' format class file (`workpackage.cls`) and a header file (`header.tex`).

The first cell of the notebook must be a `RawNBConvert` cell with at least the following contents (see the first cell of this notebook for an example):

```
\documentclass that you want to use
\input{header.tex}
whatever preamble lines you require
\begin{document}
```

Then follows whatever content you need to use your document style, e.g., font style. Write/change this file to change the front matter and appearance of the document. The script adds the `\end{document}` line after the notebook cells' contents.

The `header.tex` file provides the functionality required by the converter-created code. This content is input using the LaTeX input command. A number of pacakges are `\usepackage{}`ed, as required by the converted LaTeX code.

3.3 Company Logo

The work package template has a `logo.png` file which is used to define a logo on top of the page. The present image is empty, so no logo will show. You can add your own logo to this file.

3.4 The images directory

The script saves the png files in the `./pic` directory, and the LaTeX code also expects to find the png files there. Hence, there must be an existing `./pic` directory in the directory where you run the script. If an alternative name is given on the command line, the alternative name will be used.

3.5 Using Python code

See Listing 7.1 for the code to prepare the environment.

Listing 3.1: Caption for first listing output

```
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

The following code block has the metadata

```
{
  "listingCaption": {
    "caption": "Caption text for second listing",
    "label": "lst:listing2"
  }
}
```

which will give the code block a caption and a label, but the output is not captioned.

The code block also demonstrates how long lines are handled in this lstlistings configuration, with a linefeed symbol to mark the line.

The output of the following cell is captured in a floating listing that can be referenced. The metadata is as follows:

```
{
    "listingCaption": {
        "caption": "Caption text for second listing",
        "label": "lst:listingrad",
        "outputCaption": "Output caption text for second listing"
    }
}
```

If the meta data has an `entry and a` entry the

Listing 3.2: Output caption text for second listing

```
Radius = 6
Frontal area = 7 m2
This is a very long line of python code, that is supposed to flow over ↵
   to the next line, in order to test the listing display
```

See Listing 7.3 for the code.

```
a
b
```

```
0
1
```

The following code block has no metadata, and hence no captions on either the code or the output listings.

See Listing 7.5 for the code.

```
code with no listing caption
```

Output emanating from different cells *may* result in different output listings in LaTeX even if they appear in a single output box in the notebook. The cells below attempt to recreate a similar scenario in a different application, but it does not seem to repeat the problem here.

See Listing 7.6 for the code.

See Listing 7.7 for the code.

```
function output 1
function output 2
function output 3
9
3
```

3.6 Raw NBConvert cells

The `Raw NBConvert` cell type can be used to propagate text without any modification to the output stream. The IPython notebook will not touch this text; it is passed on as-is to the converter code. This this case we can add LaTeX code that will progress straight through to the LaTeX compiler.

This raw capability is very useful if the notebook is intended to be exported to LaTeX as primary output. I find myself writing most of such documentation in raw latex in a 'Raw NBConvert' cell. In this mode most of the notebook cells are never rendered to HTML, since you are reading and editing the LaTeX and it stays in LaTeX for display purposes as well. So you get the full benefit of all LaTeX functionality (except that it is not rendered inside the notebook) and all the other benefits of the notebook.

Raw NBConvert cell. This could be raw LaTeX as in

$$f_{WN}(\theta;\mu,\sigma)=\frac{1}{\sigma\sqrt{2\pi}}\sum_{k=-\infty}^{\infty}\exp\left[\frac{-(\theta-\mu+2\pi k)^2}{2\sigma^2}\right] \tag{3.1}$$

3.7 Captions

There are three objects that can have captions: listings, figures and tables.

Figures and tables without captions are non-floating and appear in the text at the position it is found in the input file. Figures and tables with captions are floating.

Listings are normally non-floating but can be floated (see below).

A single cell can contain any combination of captions, and these are indicated in the metadata field of the cell. It seems that the means to access the metadata changes with every new release. Try one of the following:

1. Select the `Edit Metadata` drop-down option in the `Cell Toolbar` drop-down box (towards the right of the toolbar). This should expose metadata buttons on all the cells. Click on the button and type in the metadata.
2. On the left panel click on the Property Inspector (gear icon) and open the `Advanced Tools` drop-down to expose the `Cell Metadata` text box.

Enter the meta data into the text box. Ensure that the tabs and commas are all in the right places: if so, a small tick mark will appear for you to approve the new metadata, by clicking on the tick mark. If the tabs and commas are not exactly right, the text box boundary will be red (error) and the tick mark will not be shown. If the json structure has no errors, a tick mark will appear: click on the tick mark to save the changes (as long as there are errors you cannot save).

The metadata has the following structure:

```
{
  "listingCaption": {
    "caption": "Comparison of effective drag parameters",
    "label": "lst:comparedrag",
    "outputCaption": ""
  },
  "figureCaption": {
    "caption": "Caption text for first figure",
    "label": "fig:lab1",
    "width": "0.2",
    "locator": "t"
```

```
  },
  "tableCaption": {
    "caption": "Caption text for first table",
    "label": "tab:lab1",
    "format": "{|p{10mm}|l|r|c|c|p{50mm}|p{20mm}|}",
    "fontsize": "normalsize",
    "locator": "t"
  }
}
```

Note the placement of commas. No newlines are allowed in a single string in the metadata.

3.8 Python code

Python code blocks are printed using the LaTeX `listings` package. The format details can be defined in the header template file. These code listings can be supplied with a caption and a label (but these code blocks are not floating entities).

```
  {
  "listingCaption": {
    "caption": "Caption text for first listing",
    "label": "lst:listing1",
    "outputCaption": "Caption for first listing output"
  }
}
```

where the different fields have the following meanings:

1. `caption` specifies the text to be used in the listings caption. If this metadata field is not supplied, the listing will not be given a caption.
2. `label` specifies the label to be used for the listing, which can be used in LaTeX to reference the listing. The code block uses the label exactly as given in the metadata field. If the output listing is given an `outputCaption`, the output caption is labelled by the a label starting with the `label` string given, but appended with the string `-out` (i.e., `lst:listing1-out` in this case).
3. `outputCaption` specifies the label that is used for the output listing of the cell. If this metadata field is not supplied, the output listing will not be given a caption.
4. Both the cell listing and the output listing share the same root label, but the output listing has `-out` appended to it.

We can refer to Listings 7.1, 3.2, 7.2, and 7.3.

The default is to provide colour syntax highlighting. If you don't want colour in the listings, simply remove these lines from `header.tex`

```
commentstyle=\color{mygreen},    \% comment style
keywordstyle=\color{blue},       \% keyword style
stringstyle=\color{mymauve},     \% string literal style
```

3.9 Floated code listings

In order to improve readability in the document, the code listings can be floated to the end of the document by using the `-l` switch on the command line. If this switch is used, the listing is appended to the LaTeX document and the end of the conversion. At the location in the document where the code listing would have been, a reference sentence to the code is included. The sentence is constructed as follows:

```
See Listing~\ref{lst:autolistingcellX} for the code {purpose}.
```

where the symbolic link is constructed using the cell number and the text used in the `{purpose}` part is taken from the first line of the code, if the first line is a comment with a single # followed by a space. Hence if the code in cell 3 starts with the lines

```
# to prepare the environment
import numpy as np
```

a text line of the following form will be created:

```
See Listing~\ref{lst:autolistingcell3} for the code to prepare the environment.
```

Just maintain the discipline to write good comments in the first line, keeping in mind that the comment will end up in the body of the document with the reference.

If a code cell does not start with a comment, no text will be inserted and the form will be

```
See Listing~\ref{lst:autolistingcell24} for the code.
```

If the code cell starts with a comment of the form ##, the listing will output to the tex file, but there is no text entry in the location where the code cell was. i.e., no line starting with `See Listing`...

If the code cell starts with a comment of the form ###, neither the listing nor the text entry referring to it is written to the latex file. So if you don't want the code to be shown or a reference to it made, start the first code line with ###.

3.10 Figure captions and bitmaps

PNG images, read in from externally or created by Matplotlib, are imported into the LaTeX code with lines of the following form:

```
\includegraphics[width=WIDTH\textwidth]{./pngs/test2LaTeX\_21\_0.png}
```

where the WIDTH determines the size of the image (obtained from metadata, see below), and the image file name is constructed from the notebook file name and cell information. The name is the concatenation of the notebook file name, the cell index (index number of the cell in the notebook file) and the sub-index image in the cell's output (there can be more than one image). All indices are zero based.

The width is read from the metadata, which must be structured as follows:

```
{
  "figureCaption": {
    "caption": "Caption text for first figure",
    "label": "fig:lab1",
```

FIGURE 3.1: Caption text for first figure, having a_a_a b#b%b

```
    "width": "0.2",
    "locator": "t"
  },
  "listingCaption": {
    "caption": "Listing caption in cell with a figure",
    "label": "lst:figurelisting",
    "outputCaption": "Output for a cell with a figure"
  }
}
```

Width is the fraction of LaTeX textwidth, i.e., if width=1, the graphic will be the full width of the text on the page.

The locator value is used in `\begin{figure}[tb]` or `\begin{table}[tb]` to locate the float on the page. Default value is 'tb', can be any combination of `tbhp`.

The metadata also contains a caption string, and a label for the figure. The type must be `figure`.

If the `caption` field is given (or has non-zero length), the image will be written to a LaTeX float with a caption (and label). If the `caption` fields is not given, the image will not be encapsulated in the floating figure.

If the caption string contains a substring `(###)` the substring will be replaced with the figure index (plus one) for the current cell. Hence, if there are multiple figures in the same cell, they can all use the same caption with with a running number, such as (1), (2), etc.

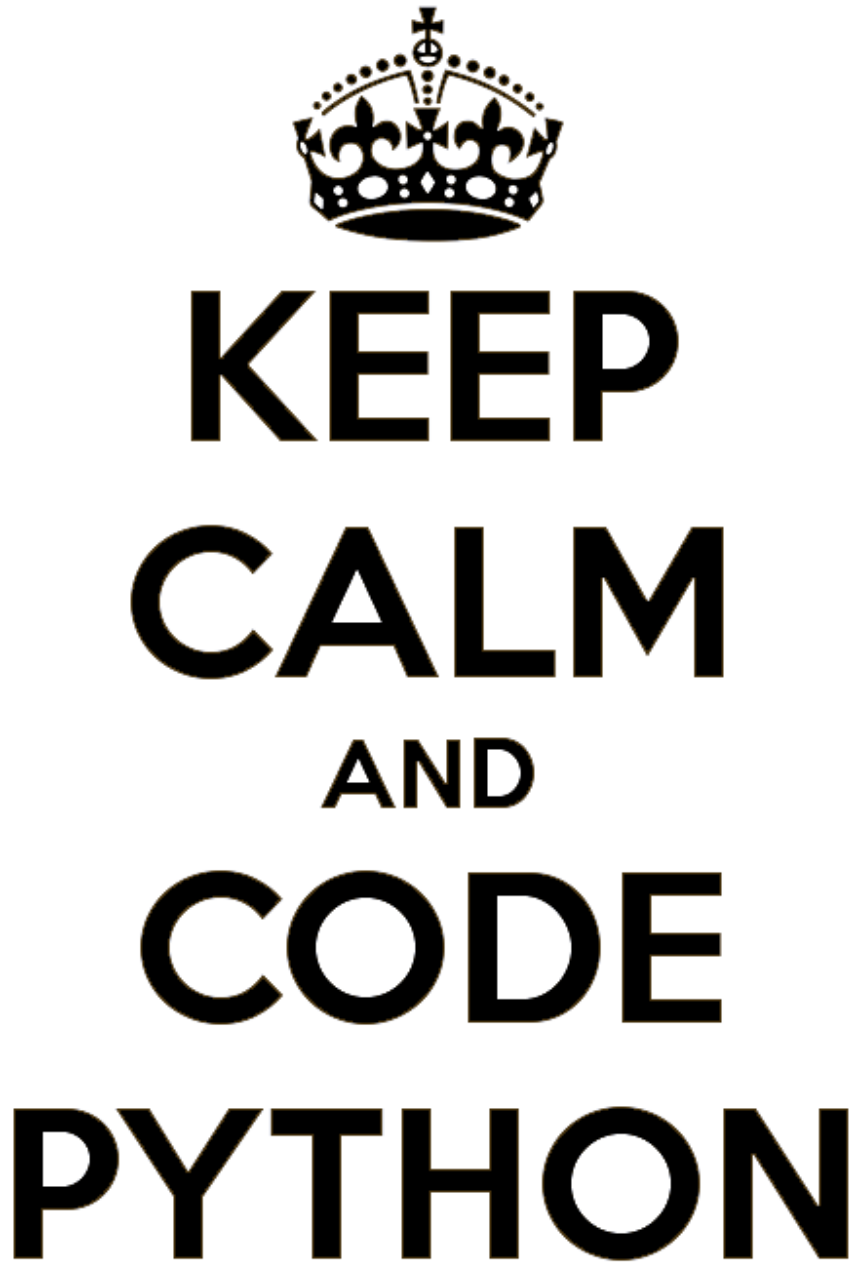
Latex special characters in the caption strings must be properly escaped as per standard LaTeX use, it is not translated or manipulated in any way. For example, underscore _ must be escaped. However, keep in mind that JSON requires backslash to be escaped itself as a double backslash in order to survive the JSON parser. So if _ is required in the final document, it must be `\\_` in the caption string. For exanple `a\\_a\\_a b\\#b\\%b` in the JSON metadata becomes `a_a_a b#b%b` in the final LaTeX caption, as shown in Figure 3.1.

See Listing 7.8 for the code this figure is given meta data caption parameters, it will be floated in the latex export.

Listing 3.3: Output for a cell with a figure

```
also force print output
```

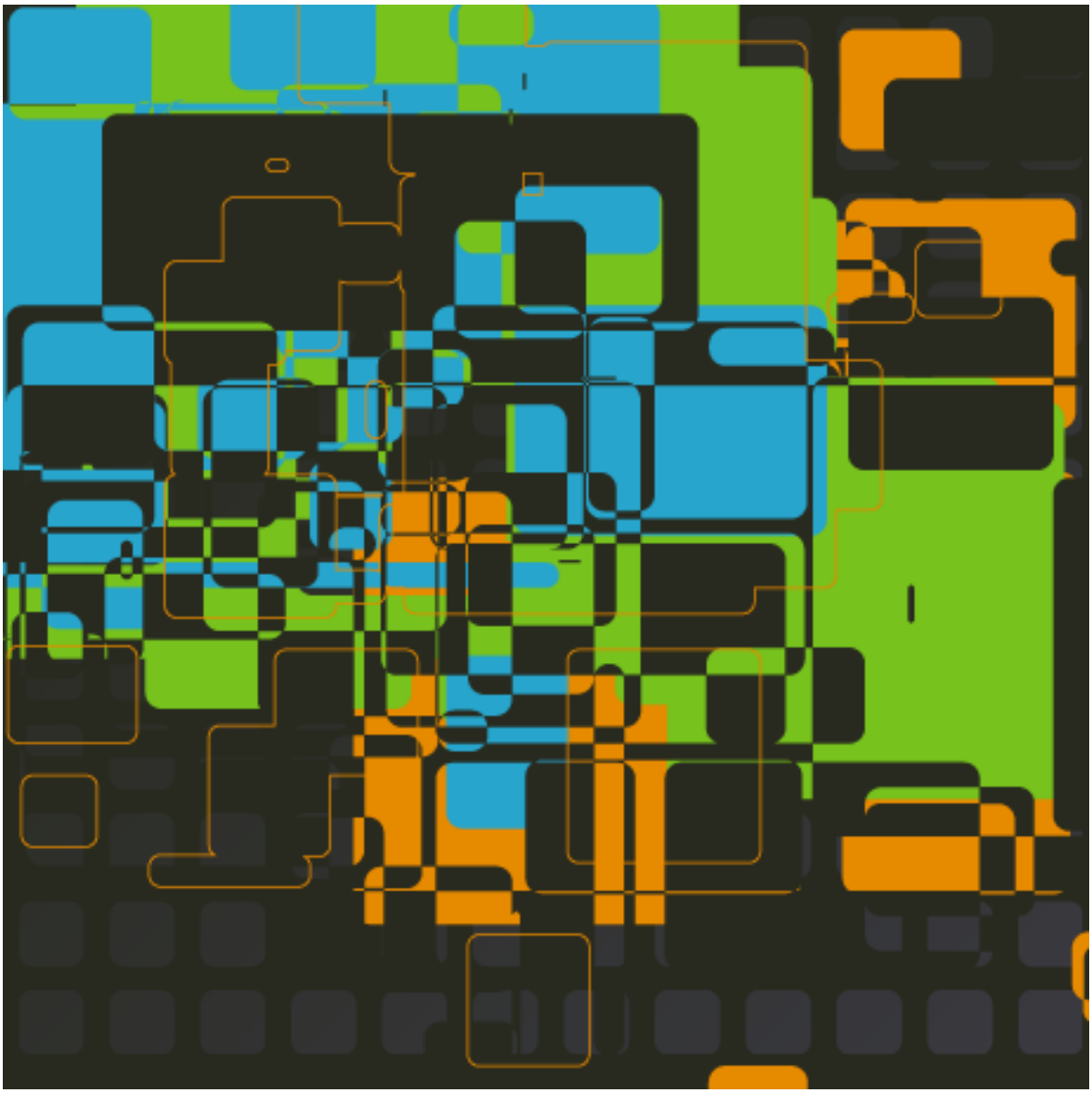
Experimenting with embedded markdown images:



The following figure is not floating, but with narrower width.

```
{
  "figureCaption": {
    "width": "0.3"
  }
}
```

See Listing 7.9 for the code this figure is not given meta data caption parameters, it will be inlined in the latex export.



The following Matplotlib plot is scaled to 40% of text width and floating, but with the 'h' locator to force the location to here.

```
{
  "figureCaption": {
    "caption": "Caption text for third figure",
    "label": "fig:lab3",
    "width": "0.4",
    "locator": "h"
  }
}
```

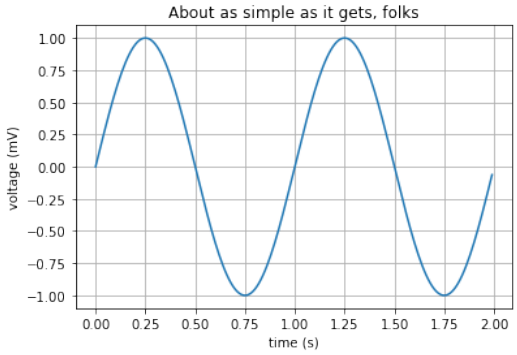


FIGURE 3.2: Caption text for third figure

JSON does not seem to like backslashes, so math embedded in the figure caption must be escaped with two backslashes, as shown in Figure 3.3.

```
{
    "figureCaption": {
        "caption": "Caption with math $\\pi$  $\\beta$, $\\rho$",
        "label": "fig:lab3math",
        "locator": "h",
        "width": "0.4"
    }
}
```

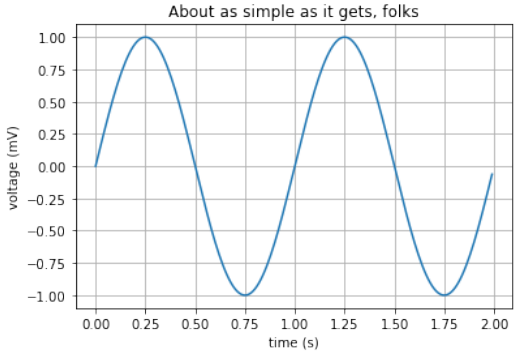


FIGURE 3.3: Caption with math π β, ρ

It is possible to have more than one plot in a single cell output. In this case each output can be made floating separately or be non-floated (you can even mix floating and non-floating). The captions are given in the metadata in the following format:

```
{
  "figureCaption": {
    "caption": "['Caption text for (first) fourth figure','',
                  'Caption text for (third) fourth figure']",
    "label": "fig:lab4",
    "width": "[0.5, 0.5, 0.5]"
  }
}
```

In this case the caption strings and scaling values are allocated to the sequence of images in the order

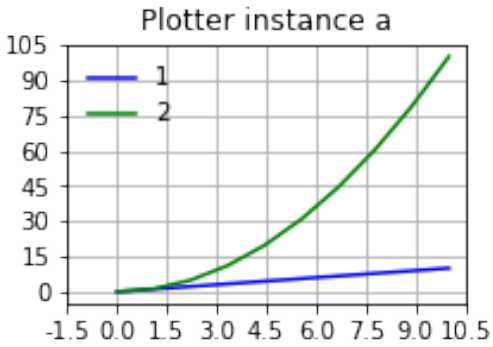


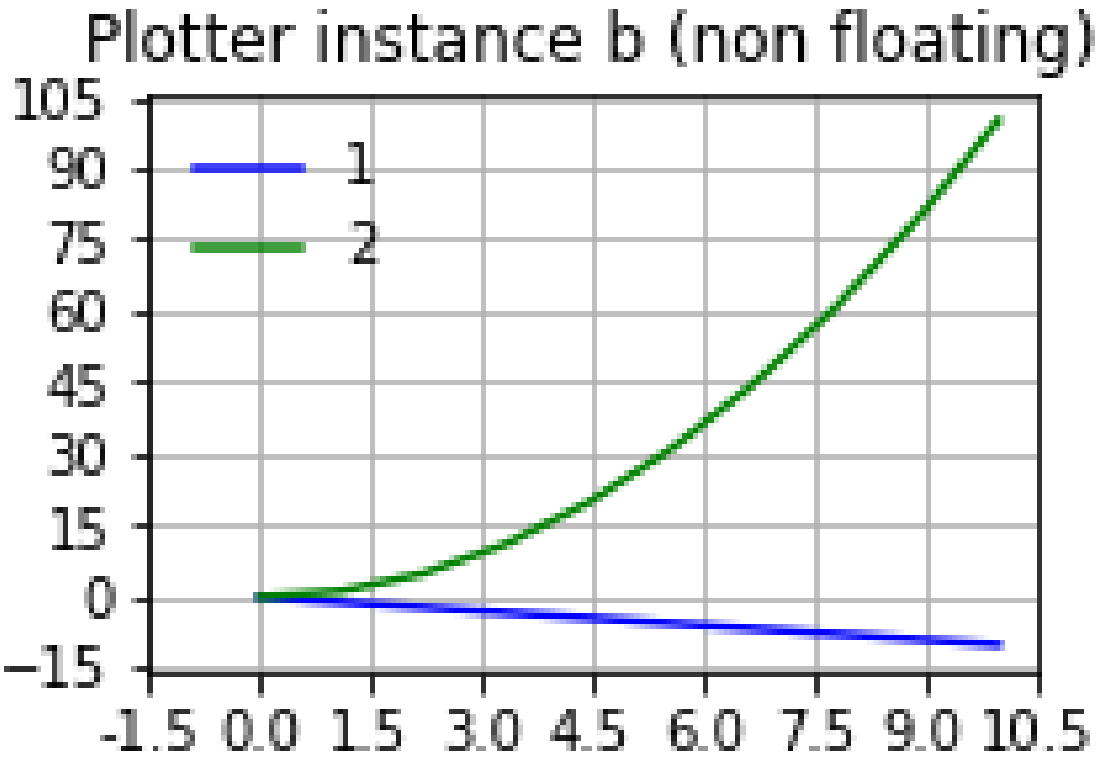
FIGURE 3.4: Caption text for (first) fourth figure

given. If a caption string is zero length, that specific image will be non-floating (as in the example of instance b here).

The float labels are determined from the root value given in the metadata, but with the output sub-index appended, as in `fig:lab4-0`, `fig:lab4-1`, `fig:lab4-2`.

The first image in a cell has two labels `fig:lab4-0` and `fig:lab4` (without the zero) so either form can be used. If you only have one figure in a cell there is no need to add the `-0`. There two references should be the same: 3.4 and 3.4 because they are both attached to the first figure in the cell.

See Listing 7.12 for the code.



To use mathematics in cells with two outputs as in the following code, the backslashes must be escaped as for a single caption, but the strings in the list must also be marked as raw strings as in `r'string'`

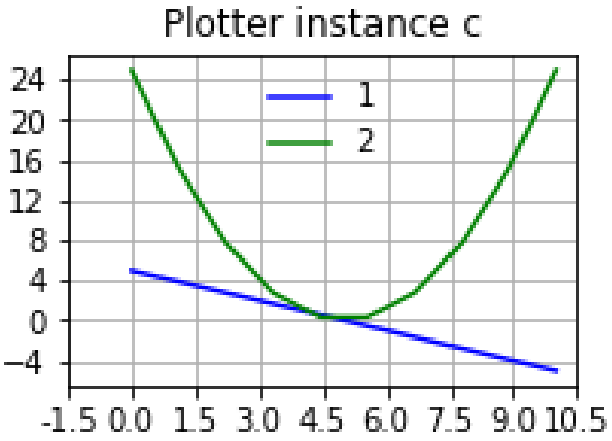


FIGURE 3.5: Caption text for (third) fourth figure

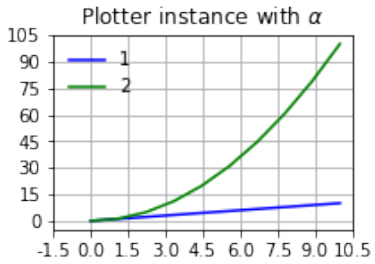


FIGURE 3.6: Caption text for α figure

```
{
    "figureCaption": {
        "caption": "[r'Caption text for $\\alpha$ figure',r'Caption ↵
           text for $\\beta$ figure']",
        "label": "fig:lab4alpha",
        "width": "[0.3, 0.5]"
    }
}
```

See Listing 7.13 for the code.

FIGURE 3.7: Caption text for β figure

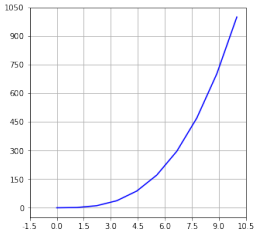


FIGURE 3.8: Shared caption (1)

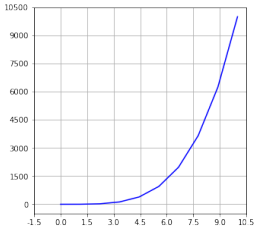


FIGURE 3.9: Shared caption (2)

Multiple pictures in the same cell can share the same caption, appended with (1), (2), etc. Use the substring（###）as a placeholder to be replaced with the running count number.

See Listing 7.14 for the code.

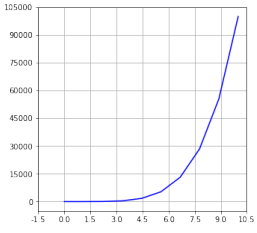
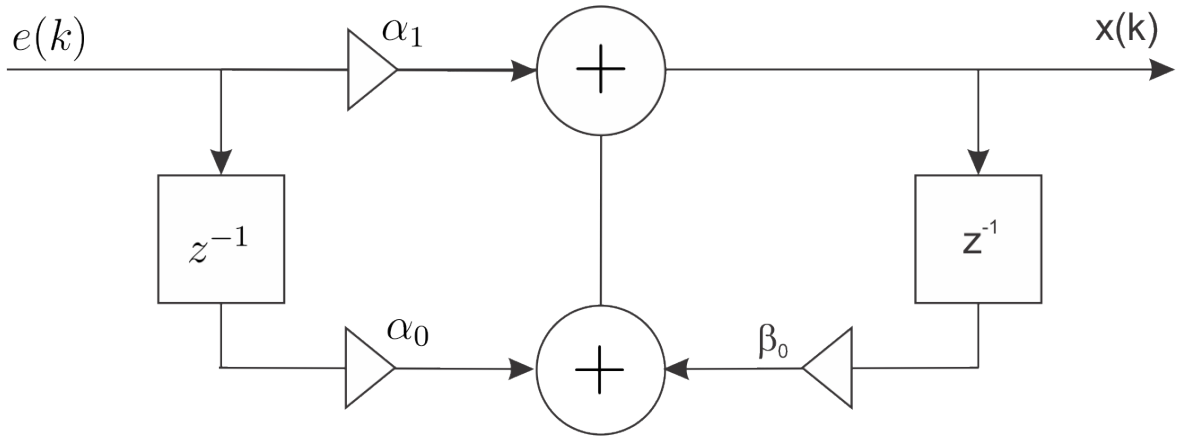


FIGURE 3.10: Shared caption (3)

You can write HTML to the cell's output and have it displayed in the browser and also in the LaTeX file.

See Listing 7.15 for the code.



There might be a picture before this text if the HTML pictures are exporting :-)

Experimenting with embedded markdown images. The figures should be Figure 3.11 and Figure 3.12.

3.11 Graphs as PDF documents

Reports with many png images tends to create large PDF files. If the graphs are exported as PDF files by the backend, the files are vector graphics and not large bitmaps, resulting in a much smaller final PDF report file.

The PDF export functionality worked well in brief testing, but it still needs more testing.

1. Set up the notebook to use the PDF backend: %config InlineBackend.figure_format = 'pdf'
2. With the PDF backend the graphs are saved as PDF documents in the notebook. This converter script will export these as PDF files to the pictures directory.
3. Compile the document with PDFLaTeX as usual. It will look for the graphics files as PDF files, not PNG files.

3.12 Embedded PDF documents

You can embed a PDF file in a notebook for display in the notebook with the following:

```
from IPython.display import IFrame
IFrame("./images/discretestrata01.pdf", width=600, height=300)
IFrame("./images/Bankedturn.pdf", width=600, height=300)
```

This currently does not work in Chrome, but it does work in Firefox, provided you set the browser to open PDF files in the browser and not download the files.

The above code displays only the second iframe in the browser, it seems to overwrite the first.

The converter will export both of the above PDF filenames for display in the latex document.

FIGURE 3.11: Keep calm 1

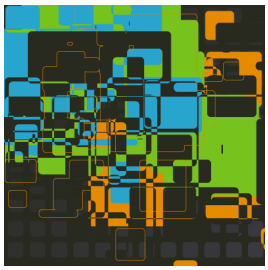


FIGURE 3.12: Keep calm 2

3.13 Cells with Python errors

The following line must be uncommented to have effect.

See Listing 7.16 for the code type(eval("this is a test string")) is not list.

3.14 Embedded code (verbatim text)

Some firewalls are set up to grant `localhost` execution rights. In this case the server can be started with the command

```
ipython notebook --ip=localhost
```

Once started, the pages are served from

```
http://localhost:8888/
```

and not from `http://127.0.0.1:8888/`.

Embedded code meant for illustration instead of execution in Python:

```
def hello\_ipython():
    print "Hello IPython!"
```

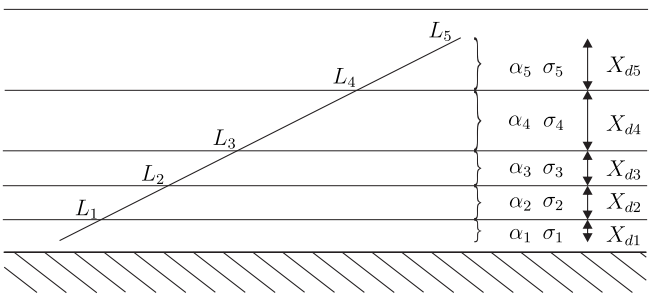


FIGURE 3.13: Stratified atmosphere

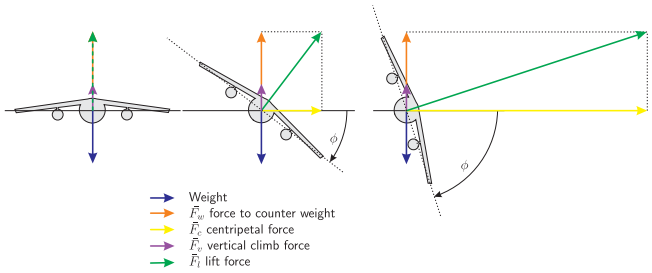


FIGURE 3.14: Banked turn

3.15 Hyperlinks, references and citations

3.15.1 Embedded hyperlinks

The IPython website[1] is the central repository of all things IPython. There are some really nice videos[2] on YouTube.

For *URI references*, the `[IPython website](http://ipython.org/)` markup structure is read and the URI is used to create a citation label (`httpipythonorg`) and the URI (`http://ipython.org/`) is written to a new LaTeX bibtex file for the references in this notebook.

For *other types of references* a workaround is required. IPython is currently weak in the area of other types of references, because of limitations in the markup language. The `ipnb2tex.py` script makes provision for including bibtex entries, by embedding the complete bibtex entry in a metadata field.

Using the above approach creates a new bibtex file using the data in the notebook. Perhaps you might want to reference an existing bibtex file, mapping to the citation labels used in the notebook. The issue here is that the IPython notebook does not have access to your LaTeX bibtex file, so it does not know the citation references you want to use (the ones in your existing bibtex file).

The approach taken here is to provide a look-up translation table in this or any prior cell of the notebook, to translate citation references from the local name to your existing bibtex name. This is done in a cross reference dictionary that maps the names created internally to the names in your external bibtex file.

The metadata has two different fields, one to do the citation label mapping and the other to embed complete bibtex entries in the metadata. The metadata field must have the following format:

```
{
  "bibxref": {
    "httpipythonorg": "httpipythonorg",
    "httpswwwyoutubecomwatchvaIXED26Wppg": "httpswwwyoutubecomwatchvaIXED26Wppg",
    "httpsenwikipediaorgwikiWrappednormaldistribution":
            "httpsenwikipediaorgwikiWrappednormaldistribution"
  },
  "bibtexentry": {
    "wing2006computational": "@article{wing2006computational,  title={Computational
          author={Wing, Jeannette M},  journal={Communications of the ACM},  volu
          number={3},  pages={33--35},  year={2006}}",
    "gracec": "@MISC{gracec,  author = {Grace Cathedral},  title = {The Cathedral
          url = {http://www.gracecathedral.org/labyrinth/}}"
  }
```

```
}
```

where (1) the bibxref keys are the local names and the values are the names in your existing bibtex file and (2) the bibtexentry keys are the citation labels used in the notebook. In the above example the bibxref maps to the same names, because I am using the locally generated bibtex file. Normally you would use your bibtex database entries.

Note that the IPython notebook is somewhat finicky on the json format; each single entry in the above metadata must be on a single line (line-feeds inside the strings are not allowed — lines tend to be very long). Also, note the location of commas, there should be commas after all entries, except the last entry in a given scope.

The `ipnb2tex.py` script:

1. Loads/appends the bibxref translation table from any/all cells (if present).
2. Reads the `[]()` markup structure and then builds a citation reference from the URI (by removing some characters).
3. Create a bibtex entry for the reference (subsequently written to file).
4. Using bibxref, translates the local citation label to your existing citation label.

If you included bibtex items in the metadata, you can refer to them using the normal LaTeX notation. Test [3] and [4].

At the conclusion of the processing the script reads all existing `*.bib` files in the cirrent folder and combine all of them into one file. This way you can include prior existing bib files.

3.16 General markdown formatting

Markdown basics: lists, markup and code

- list item1 in markdown format.
- list item2
  - nested list item3 - font attributes not yet supported.
- *italics*
- **bold**
- `fixed font`

1. Enumerated list item 1 in markdown format.
   (a) sub lit element 1
   (b) sub lit element 2
2. Enumerated list item 2.

FIGURE 3.15: Boxy picture

The markup language (or the converter) breaks if an itemized list and an enumerated list are immediately adjacent — we need to separate the lists by text with with a <p/>.

Please note that the converter does not currently allow free standing (enter-twice) paragraphs inside lists. Please make the paragraphs touch and use double spaces at the end to force a new line. This will be converted to loose standing paragraphs in LaTeX.

1. list item1 in markdown format.
2. list item2

   item2 new parra 1

   item2 new parra 1
3. list item3

   item3 new parra 1

   item3 new parra 2
4. list item4 [this does not work correctly]

   list item4 [this does not work correctly]

   item4 new parra 1 [this does not work correctly]

   list item4 [this does not work correctly]

   list item4 [this does not work correctly]

   item4 new parra 1 [this does not work correctly]

   item4 new parra 2 [this does not work correctly]
5. list item5 [this should work correctly]

   item5 new parra 1 [this should work correctly]. We reference Figure 3.15 here.

   item5 new parra 2 [this should work correctly]

Lists in HTML format

- list item
- list item
  - nested list item
- *italics*
- **bold**
- `fixed font`

TABLE 3.1: Caption text for first table

a	b			c			1
e	f	g		h	i		2
	j		k	l	m		3
n	o	p	q		r		4
s	t	u	v	w	x		5

1. Enumerated list item 1.
    (a) sub lit element 1
    (b) sub lit element 2
2. Enumerated list item 2.

3.17 Tables

The table in this cell is rendered in LaTeX with the following metadata:

```
{
  "tableCaption": {
    "caption": "Caption text for first table",
    "label": "tab:lab1",
    "format": "{|p{10mm}|l|r|c|c|p{50mm}|p{20mm}|}",
    "fontsize": "normalsize",
    "locator": "t"
  }
}
```

A complex HTML table with row spans and column spans:

The tables in this cell are rendered in LaTeX with the following metadata:

```
{
  "tableCaption": {
    "caption": "['Caption text for (first) second table','','Caption text for (thi
    "label": "tab:lab2",
    "format": "['{|p{20mm}|r|}','','{|c|l|}']",
    "fontsize": "['normalsize', 'tiny', 'Large']"
  }
}
```

Github flavoured markdown tables are supported in the IPython notebook (floating in LaTeX as Table 3.2):

This	is
a	small table

second table (non-floating in LaTeX):

TABLE 3.2: Caption text for (first) second table

This	is
a	table

TABLE 3.3: Caption text for (third) second table

First Header	Second Header
Content Cell	Content Cell
Content Cell	Content Cell

This	is
a	small table

First Header	Second Header
Content Cell	Content Cell
Content Cell	Content Cell

PHP Markdown Extra is also supported (floating in LaTeX as Table 3.3):

First Header	Second Header
Content Cell	Content Cell
Content Cell	Content Cell

Both of these markup extensions require Python Markdown 2.4.1 to render HTML.

Table 3.4 has an hierarchical structure on both columns and rows. The percentage and underline characters are also preserved.

The infrared sensor example has the following design characteristics:

TABLE 3.4: Caption text for hierarchical table

		CHead 1	CLevel 1:1	
		CHead 2	CLevel 2:1	CLevel 2:2
RHead 1	RHead 2	RHead 3		
RLevel 1:1%_	RLevel 2:1	A	6000%	6156
		B	2417	2471
		C	1274	1347
	RLevel 2:2	A	10909	11041
		B	4400	4408
		C	2309	2319
	RLevel 2:3	A	11573	11178
		B	4461	4976
		C	2432	2410
RLevel 1:2%_	RLevel 2:1	A	6728	6595
		B	2322	2679
		C	1300	1474
	RLevel 2:2	A	1210	12344
		B	4845	4367
		C	2618	2525
	RLevel 2:3	A	12553	12117
		B	4895	4835
		C	2566	2991

Characteristic	Value	Unit	Motivation
Spectral response	3.7–4.9	μm	detector specification
Pixel size (x and y)	12	μm	detector specification
Pixel fill factor	0.95		detector specification
Detector temperature	80	K	detector specification
Detector external quantum efficiency	0.8		detector specification
Detector internal quantum efficiency	0.75		detector specification
Number rows	144	-	detector specification
Number columns	256	-	detector specification
Detector PRNU stddev	0.2		detector specification
Well capacity at 1 V	3.2×10^6	e	detector specification
Sense node voltage	3.0 → 1.0	V	detector specification
F-number	3.2	-	detector specification
Band gap 0 K	0.235	eV	material property
Varshni A	0.00068		material model
Varshni B	500		material model
Dark FOM	4×10^{-9}	nA/cm^2	material model
Dark cm	1		material model
Dark FPN stddev	0.4		material model
Well capacitance at 1.0 V	5.13×10^{-13}	F	by calculation
k1	5.13×10^{-13}	CV	by calculation
Gain at 1.0 V	3.125×10^{-7}	V/e	by calculation
Pixel IFOV (x and y)	100.0×10^{-6}	rad	design choice
Frame time	0.02	s	design choice
Focal length	0.12	m	design choice
Full field angle	0.84	deg	focal length and detector

Charge well (sense node) capacitance $C = nq/V$. The charge well is filled to capacity at the minimum sense node voltage $C = 3.2 \times 10^6 \times 1.6 \times 10^{-19}/1.0 = 0.513 \times 10^{-12}$ F. Then $k_1 = CV = 0.513 \times 10^{-12} \times 1 = 0.513 \times 10^{-12}$.

Sense node gain is given by $V/n = q/C = 1.6 \times 10^{-19}/0.513 \times 10^{-12} = 3.12 \times 10^{-7}$ V/e.

A table can be formatted with the metadata, even if the table is not floating with a caption. For example, the following table only uses the following metadata:

```
{
  "tableCaption": {
    "format": "{|p{10mm}|p{50mm}|p{100mm}|}",
    "fontsize": "footnotesize"
  }
}
```

Column Number	Column Description	Example Values
1	Unique row identifier	MOD15A2.A2000057.h12v03.004.2002357024124.FparExtra_QC MOD15A2.A2000057.h12v03.004.2002357024124.Lai_1km
2	MODIS Land Product Code	MOD15A2
3	MODIS Acquisition Date [A (YYYYDDD)]	A2000057
4	User selected center point coordinates and specified width (Samp) and height (Line) of bounding rectangle in pixels. Width x height denotes number of Product values starting in Column 7. (e.g., 7 x 7 = 49)	Lat55.879620Lon-98.480810Samp7Line7
5	MODIS Processing Date (YYYYDDDHHMMSS)	2002357024124
6	Product Scientific Data Set (Band): Indicates type of values to follow. Specific values vary by Product. Data quality information are interleaved.	MOD15A2: FparExtra_QC, FparLai_QC, Fpar_1kmMOD17A2: Gpp_1km, PsnNet_1km, Psn_QC_1km
7 to N	Data values of type as specified. Number of data columns as given in Column 4. Definition of QC component values vary by Scientific Data Set.	QC: 00100001, 01100001, 01100001, ...Measurement: 2, 2, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, ...to N

Pandas dataframe HTML tables are also rendered.

See Listing 7.17 for the code.

	one	two
a	1.0	1.0
b	2.0	2.0
c	3.0	3.0
d	NaN	4.0

3.17.1 Font size in tables

If both `latex/fontsize` and `tableCaption/fontsize` are present then the `tableCaption/fontsize` takes preference, and the table (only) is rendered in the specified fontsize. The floating caption is in the normal document font size.

```
{
  "latex": {
    "fontsize": "scriptsize"
  },
  "tableCaption": {
    "caption": "Caption text for Pandas table",
    "label": "tab:lab1",
    "format": "somelabel",
    "fontsize": "tiny"
  }
}
```

4 Embedding LaTeX code in output cells

Jupyer uses MathJax for LaTeX rendering in the markdown cells. This is very useful for documentation in general, but sometimes MathJax is not sufficient for special LaTeX constructs. LaTeX constructs can be embedded (but not rendered in Jupyter) in output cells. The LaTeX in the output cell can then be rendered/typeset when the converted LaTeX document is built.

Any string you construct in a code cell and then use the `IPython.display.Latex` function to display will be embedded in the output cell as a data of mime type `text/latex`. The ipynb2tex converter will then process this LaTeX in the output cell as regular LaTeX in the target document. So the following code:

```
from IPython.display import Latex
lstr = 'This is a \LaTeX string.'
Latex(lstr)
```

Will make a mime type `text/latex` data entry with the contents of `lstr` in the output cell. When rendered in the notebook, it will simply show the text as entered.

With this functionality, LaTeX code of any complexity can be constructed by code, for eventual typesetting by the LaTeX compiler after the notebook has been converted to LaTeX.

4.1 Pandas DataFrame to LaTeX

The following example shows how to export a Pandas dataframe to a `IPython.core.display.Latex object` in the string `lstr`. When viewed in the notebook, the table is shown as a LaTeX code, but when the conversion to a LaTeX document takes place, the LaTeX is rendered as regular LaTeX in the output PDF file.

The table format specification is provided by the Pandas to_latex(column_format)[5] argument, as in

`'|l|l|p{50mm}|c|r|'`

See Listing 7.18 for the code to create the dataframe and write the output to LaTeX object at default font size.

	A	B	C	D	E
2013-01-01	0.083049	-0.106439	0.381134	-0.518548	-1.255566
2013-01-02	-0.570557	-0.456681	1.154854	-0.250722	0.825798
2013-01-03	-1.683748	1.430985	-0.318301	0.720622	0.543044

See Listing 7.19 for the code.

A	B	C	D	E
0.083049	-0.106439	0.381134	-0.518548	-1.255566
-0.570557	-0.456681	1.154854	-0.250722	0.825798
-1.683748	1.430985	-0.318301	0.720622	0.543044

Set the font size for the typesetting by using the `latex/fontsize` setting in the cell json metadata. Use the regular LaTeX font size definitions but without the backslash (Huge, huge, LARGE, Large, large, normalsize, small, footnotesize, scriptsize, tiny):

TABLE 4.1: Caption text for the Pandas table

	A	B	C	D	E
2013-01-01	0.083049	-0.106439	0.381134	-0.518548	-1.255566
2013-01-02	-0.570557	-0.456681	1.154854	-0.250722	0.825798
2013-01-03	-1.683748	1.430985	-0.318301	0.720622	0.543044

```
{
  "latex": {
    "fontsize": "LARGE"
  }
}
```

Of course you may have to use this setting in conjunction with any other json settings.

See Listing 7.20 for the code to write the table in LARGE.

	A	B	C	D
2013-01-01	0.083049	-0.106439	0.381134	-0.518548
2013-01-02	-0.570557	-0.456681	1.154854	-0.250722
2013-01-03	-1.683748	1.430985	-0.318301	0.720622

A Pandas dataframe exported with `IPython.core.display.Latex object` can also be typeset as a floating table by setting the usual table metadata for the floating table. The table format specification is provided by the Pandas `.to\_latex(column\_format)` argument, so the `tableCaption/format` specification in the json is ignored.

```
{
  "tableCaption": {
    "caption": "Caption text for Pandas table",
    "label": "tab:lab1",
    "format": "IGNORED because this is a pandas dataframe export!!",
    "fontsize": "tiny"
  }
}
```

See Listing 7.21 for the code to write a floating table in tiny font size.

4.2 Other LaTeX exports

Write latex strings

Samples per cluster ID:
6 7 8 9 5

Samples per visual material:
4 4 7 2 2 2 4 2 2 5

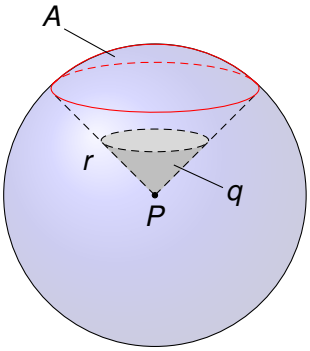
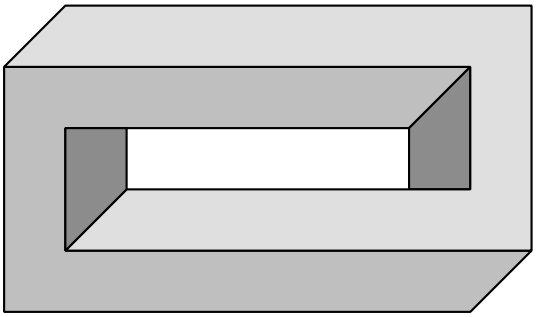


FIGURE 4.1: Caption text for tikz figure

See Listing 7.22 for the code display a tikz graphic.



See Listing 7.23 for the code display another tikz graphic, as a floating figure.

The next example creates a LaTeX string and outputs it as a Latex object. The code calculates rules with increasing heights, with this code:

```
lstr = '\n\nThis paragraph is embedded as \LaTeX{} in a Jupyter output cell. '\
            + 'One day this could be a complex computed \LaTeX{} string.  '\
            + 'This example only has rising rules...  \n'
for height in range(1,20):
    lstr += '\\rule[0pt]{{10pt}}{}{}pt{} '.format('{',height,'}')
Latex(lstr + '\n')
```

This paragraph is embedded as LaTeX in a Jupyter output cell. One day this could be a complex computed LaTeX string. This example only has rising rules...

Jelte Fennema wrote pylatex[6] to create LaTeX text using Python code. He wrote the code to create complete documents, but also to write fragments[7] — which we can use here: "Classes can be part of a single document, or can act as pieces on their own. With the `dumps\_as\_content` method, most classes can return their LaTeX-formatted code, and with the `generate\_tex` method, this code can be written to a file."

I only spent limited time on pylatex and got the following example to work (it requires `pip install pylatex`).

See Listing 7.25 for the code to format a numy array in LaTeX.

$$arr = \begin{bmatrix} -1.5238753267681426 & -0.054106597608410884 & 0.3133992734699635 & -1.914463011081044 \\ -0.9730171110025135 & 1.2607379010435467 & 0.9708551354916619 & -1.5426473382692332 \\ 1.5617776961021184 & 1.6862635883897612 & -1.187051258257019 & -0.6077778495302126 \\ -0.7381209385420484 & -0.6953658326619776 & -0.3490201919937157 & -0.1781985647847829 \\ 1.5326454450945042 & 0.6131410607502967 & -0.5031807629333506 & 0.5286306282646915 \\ -0.8970203037261731 & -0.8958130486496836 & 0.040453221245678746 & -0.47384738447189984 \end{bmatrix}$$

5 Math

Inline math delineated with single dollar symbols are rendered inline with paragraph text, as in $\alpha+\beta=\gamma$.

Using math mode (anything between two dollar symbols) is interpreted as LaTeX math:

$$D_{KL}(P||Q)=\sum_{i} ln(\frac{P(i)}{Q(i)})P(i). \tag{5.1}$$

$$\begin{aligned} E=\tfrac{\phi}{A_o}=\int [& \\ & \tau_a(R)\tau_o \\ & [\\ & \qquad \Omega_t k_i\left(\epsilon_t L(T_t)\right) && [1] \\ & \qquad +\Omega_t k_i(1-\epsilon_t)\left(\tfrac{\rho E_{\text{sun}}}{\pi}\right) && [2] \\ & \qquad +\Omega_t(1-k_i)(1-\epsilon_t)\left(\tfrac{\rho E_{\text{sun}}}{\pi}\right) && [3] \\ & \qquad +(\Omega_p-\Omega_t)\left(\tfrac{\rho E_{\text{sun}}}{\pi}\right) && [4] \\ &] \\ & +\tau_o L_{\text{path}}(R)\Omega_p && [5] \\ &]\,d\lambda \end{aligned} \tag{5.2}$$

The probability density function for yaw angle is the wrapped normal distribution[8]. The first form uses two dollar symbols to show the equation in LaTeX display math:

$$f_{WN}(\theta;\mu,\sigma)=\frac{1}{\sigma\sqrt{2\pi}}\sum_{k=-\infty}^{\infty}\exp\left[\frac{-(\theta-\mu+2\pi k)^2}{2\sigma^2}\right] \tag{5.3}$$

The second example uses `begin{equation}` and `end{equation}`:

$$f_{WN}(\theta;\mu,\sigma)=\frac{1}{\sigma\sqrt{2\pi}}\sum_{k=-\infty}^{\infty}\exp\left[\frac{-(\theta-\mu+2\pi k)^2}{2\sigma^2}\right] \tag{5.4}$$

$$X_{WN}(\theta;\mu,\sigma)=\tfrac{1}{\sigma\sqrt{2\pi}}\textstyle\sum_{k=-\infty}^{\infty}\exp\left[\frac{-(\theta-\mu+2\pi k)^2}{2\sigma^2}\right] \tag{5.5}$$

6 General LaTeX Crib Notes

This chapter contains general notes, not necessarily pertaining to the Jupyter notebook converter.

6.1 Quotes

Use `\verb`word'+ [+word'] instead of `'word'` ['word']. Likewise `\verb`word"'+ [+word"] instead of `''word''` ["word"].

6.2 Degree Symbol

The degree symbol in LaTeX should not be done with `\circ` as shown here: 45°. Rather use `\usepackage{gensymb}` and then `45\textdegree{}` 45°, or `17~\celsius` 17 ℃. Better yet, use the siunitx package described below.

6.3 Test Sub- and Superscripts

Create superscripts and subscripts in body text with `m\textsuperscript{2}` and `CO\textsubscript{2}` to get m^2 and CO_2, instead of m^2 or CO_2.

6.4 siunitx

Load `siunitx` package for some really nice features. Load it with `\usepackage[detect-weight,detect-m`

LaTeX	Output
\num {3e-5}	3×10^{-5}
\si {\metre \per \second \squared }	$\mathrm{m\,s^{-2}}$
\si {\micro \metre }	µm upright, instead of μm
\SIlist {0.13;0.67;0.80}{\milli \metre }	0.13 mm, 0.67 mm and 0.80 mm
\si {\metre \per \second \squared }	$\mathrm{m\,s^{-2}}$
\si {kg.m.s^{-1}}	$\mathrm{kg\,m\,s^{-1}}$
\si {\kilogram \metre \per \second }	$\mathrm{kg\,m\,s^{-1}}$
\si [per-mode=symbol]{\kilogram \metre \per \second }	kg m/s
\si [per-mode=symbol]{\kilogram \metre \per \ampere \per \second }	kg m/(A s)
\si [per-mode=reciprocal]{\kilogram \metre \per \ampere \per \second }	$\mathrm{kg\,m\,A^{-1}\,s^{-1}}$
\si {\kilogram \metre \per \ampere \per \second }	$\mathrm{kg\,m\,A^{-1}\,s^{-1}}$
\si [per-mode=fraction]{\kilogram \metre \per \ampere \per \second }	$\frac{\mathrm{kg\,m}}{\mathrm{A\,s}}$
\si [per-mode=symbol]{\watt \per \metre \squared \per \steradian }	W/(m^2 sr)
\ang {12.3}	12.3°
\ang {1;2;3}	1°2′3″
\si {\farad \squared \lumen \candela }	$\mathrm{F^2\,lm\,cd}$
\si [inter-unit-product = \ensuremath {{}\cdot {}}]{\farad \squared \lumen \candela }	$\mathrm{F^2 \cdot lm \cdot cd}$

6.5 Math functions

Math functions should be entered thus: `\tan()` to yield $\tan()$ rather than `tan()` to yield $tan()$.

6.6 Use of itemize and enumerate

Use `itemize` and `enumerate` only when there is a sense of counting or a need to have an explicit list where the difference between the entries must be emphasised. Don't use these lists when a simple sentence would suffice.

For example, the following would work better in a normal sentence and not as an 'itemize': The following types of sensors are commonly used for kinematic recordings:

- accelerometers
- gyroscopes
- Hall sensors (for static magnetic fields)
- induction sensors (for dynamic magnetic fields)

Instead write: Sensors such as accelerometers, gyroscopes, Hall sensors (for static magnetic fields), and induction sensors (for dynamic magnetic fields) are commonly used for kinematic recordings.

Obey the grammar rules for punctuation in these types of lists. If each entry is a separate sentence, start with a capital letter and end with a full stop. If the sentence spreads across the list use commas

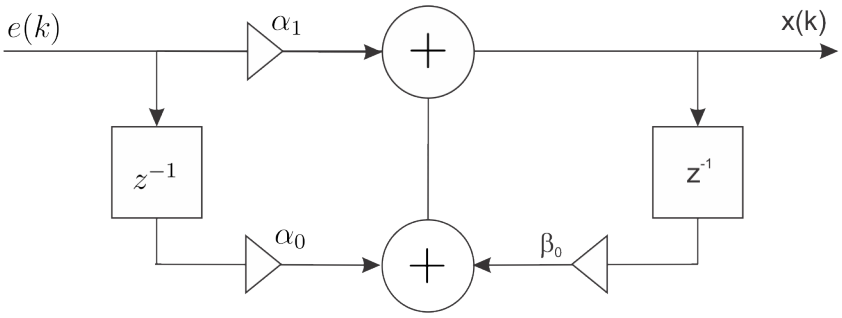
and capital letters appropriately, as follows: The following types of sensors are commonly used for kinematic recordings:

- accelerometers,
- gyroscopes,
- Hall sensors (for static magnetic fields), and
- induction sensors (for dynamic magnetic fields).

6.7 LaTeX fonts in Figures

If at all possible use LaTeX fonts in figures. For example in the following figure, the fonts on the left side fits better with the equation

$$x(k) = \alpha_1 e(k) + \alpha_0 e(k-1) - \beta_0 x(k-1) \tag{6.1}$$



6.8 Fonts, Emphasis and Shouting

Avoid CAPITALISATION OF WORDS. It is like SHOUTING!

Avoid underlined words, unless there it is really necessary. It is like shouting when normal voice is adequate.

Use emphasis to *emphasise words that must stand out*. Do not use it too often, perhaps less than three to five times per page. Emphasis is like a *waving flag*. If there are *too many flags* waving on a page, it *confuses the reader*.

Use `verbatim` to highlight software-related words, such as variable names, module/package/library names, command line text, program output, etc. The user associates verbatim font with source code listings, so use verbatim with software 'words'.

6.9 Math Conventions

The IUPAC Green Book, `https://en.wikipedia.org/wiki/IUPAC\_book`, conveniently summarises international standards in formatting mathematics text and SI units. See also `www.tug.org/TUGboat↵/Articles/tb18-1/tb54becc.pdf` and `https://www.iso.org/standard/31887.html`.

The mathematical differential operator must be [ISO 80000-2:2009(E), 2-11.12] upright, not cursive (although pure mathematicians and some house styles prefer cursive).

```
\usepackage{commath}
\newcommand{\der}{\operatorname{d!}{}}
$(\der{}x)$  or $(\dif{}x)$ instead of $(dx)$
```

$(\mathrm{d}!\,x)$ or $(\mathrm{d}x)$ instead of $(d\,x)$

```
\usepackage{commath}
\begin{equation*}F(t) = m \od[2]{x(t)}{t}\end{equation*}
```

$$F(t) = m\frac{\mathrm{d}^2x(t)}{\mathrm{d}t^2}$$

Mathematical variables must be cursive ($\alpha\beta$), but constants must be upright ($\mathrm{\pi}$) not cursive (π). The code

```
\usepackage[utopia]{mathdesign}
\usepackage[OMLmathrm,OMLmathbf]{isomath}
\newcommand\ct[1]{\text{\rmfamily\upshape #1}}
cursive: $\pi$  $\mathbfit{\pi}$ or upright $\mathrm{\pi}$ $\mathbf{\pi↵
   }$\
or\
cursive: $e^{i\pi}-1=0$ or upright $\ct{e}^{\ct{i}\piup}-1=0$.
```

yields cursive: π $\boldsymbol{\pi}$ or upright $\mathrm{\pi}$ $\mathbf{\pi}$ or cursive: $e^{i\pi}-1=0$ or upright $\mathrm{e}^{\mathrm{i}\pi}-1=0$.

Variable subscripts should be typeset in roman if the subscript is a recognisable word: v_{aircraft} instead of $v_{aircraft}$

6.10 Centering in Floats

Use `\centering` in floats instead of `\begin{center}\end{center}`, as in

```
\begin{figure}
\centering
\includegraphics[width=0.1\textwidth]{./images/keep-calm-and-code-python\_BW.png}
\caption{Keep calm and code Python\label{fig:keep-calm-and-code-python\_BW}}
\end{figure}
```

6.11 Forcing Floats to Show

The floats accumulate up to a point where some or all of these floats are processed an displayed on a page. The floats are displayed in the order given in the input file. The rules as to when the float are processed and displayed are not entirely clear to me. LaTeX appears to have a limit of around 18 unprocessed floats, beyond which an error message is given.

You can force all unprocessed floats to output in a number of different ways:

1. Use the \clearpage command. This command immediately stops any further text output and displays all unprocessed floats in the order given. The problem with this command is that no text is output until all floats are output. This means that if the \clearpage command is near the top of a page, the rest of the page could be empty (unless a float fits there).

2. The \afterpage{} command processes whatever command you have given in, after the text on the current page is processed and output. So by using an \afterpage{\clearpage} command at the point where you want to force float output, further text is processed on the current page, filling and outputting the current page, before the \clearpage command is issued: no more half-empty pages!

3. The \FloatBarrier command causes all unprocessed floats to be processed immediately. Unlike \clearpage, it does not start a new page. The command requires the placeins package.

4. To keep floats in the sections in which they were included, use \usepackage[section]{placeins}.

5. If you don't want your floats to float, you can use the [H] float specifier from the float package.

6. If you don't want your figure to float, don't use a floating environment; you can use, for example, a center environment and (if a caption is needed) the \captionof command provided by the capt-of (or caption) package. http://tex.stackexchange.com/questions/32598/force-latex-image-to-appear-in-the-section-in-which-its-declared

```
\usepackage{capt-of}
\centering
  \includegraphics{foo}
  \captionof{figure}{A non floating figure}
  \label{fig:test}
```

6.12 Python and module versions, and dates

```
The watermark extension is already loaded. To reload it, use:
  %reload_ext watermark
Python implementation: CPython
Python version       : 3.8.3
IPython version      : 7.19.0

numpy : 1.19.2
scipy : 1.5.2
pyradi: 1.1.2

Compiler    : MSC v.1916 64 bit (AMD64)
OS          : Windows
Release     : 10
Machine     : AMD64
Processor   : Intel64 Family 6 Model 165 Stepping 2, GenuineIntel
CPU cores   : 16
Architecture: 64bit

Git hash: 25292a7d413d121bb65a59ebb983e2a32dad9323
```

7 Listings

Listing 7.1: Caption text for first listing

```
# to prepare the environment
from IPython.display import display
from IPython.display import Image
from IPython.display import HTML

import os.path
import numpy as np
%matplotlib inline

import this
```

Listing 7.2: Caption text for second listing

```
#
print(' ')
print('Radius = {}'.format(6))
print('Frontal area = {} m2'.format(7))
print('This is a very long line of python code, that is supposed to ↵
   flow over to the next line, in order to test the listing display')
```

Listing 7.3: Caption text for third listing

```
# to print the value
a = ['a','b']
for v in a:
    print(v)
```

Listing 7.4: Code Listing in cell 28

```
## this should not appear in the body of the doc if float listings
a = [0, 1]
for v in a:
    print(v)
```

Listing 7.5: Code Listing in cell 30

```
print('code with no listing caption')
```

Listing 7.6: Code Listing in cell 32

```
def doprint(i):
    print('function output {}'.format(i))
    return int(i)
```

Listing 7.7: Code Listing in cell 33

```
for i in [1,2,3]:
    si = str(i)
    v = doprint(si)

val = v * 3
print(val)
print(v)
```

Listing 7.8: Listing caption in cell with a figure

```
#this figure is given meta data caption parameters, it will be floated ↵
   in the latex export
display(Image(filename='images/keep-calm-and-code-python_BW.png', width↵
   =250, height=250))
print('also force print output')
```

Listing 7.9: Code Listing in cell 46

```
#this figure is not given meta data caption parameters, it will be ↵
   inlined in the latex export
display(Image(filename='images/random-squares-2.png', width=200, height↵
   =200))
```

Listing 7.10: Code Listing in cell 48

```
##
%matplotlib inline

import pylab as pl
import numpy as np

t = np.arange(0.0, 2.0, 0.01)
s = np.sin(2*np.pi*t)
pl.plot(t, s)
pl.xlabel('time (s)')
pl.ylabel('voltage (mV)')
pl.title('About as simple as it gets, folks')
pl.grid(True)
```

Listing 7.11: Code Listing in cell 52

```
##
%matplotlib inline

import pylab as pl
import numpy as np

t = np.arange(0.0, 2.0, 0.01)
s = np.sin(2*np.pi*t)
pl.plot(t, s)
pl.xlabel('time (s)')
pl.ylabel('voltage (mV)')
pl.title('About as simple as it gets, folks')
pl.grid(True)
```

Listing 7.12: Code Listing in cell 54

```
import pyradi.ryplot as ryplot

x = np.linspace(0,10,10)
a = ryplot.Plotter(1,figsize=(3,2))
b = ryplot.Plotter(2,figsize=(3,2))
c = ryplot.Plotter(3,figsize=(3,2))
for i in [1,2]:
    a.plot(1,x,x ** i,'Plotter instance a',label=[str(i)])
    b.plot(1,x,(-x) ** i,'Plotter instance b (non floating)',label=[str↵
       (i)])
    c.plot(1,x,(5-x) ** i,'Plotter instance c',label=[str(i)])
```

Listing 7.13: Code Listing in cell 57

```
import pyradi.ryplot as ryplot

x = np.linspace(0,10,10)
a = ryplot.Plotter(1,figsize=(3,2))
b = ryplot.Plotter(2,figsize=(3,2))
for i in [1,2]:
    a.plot(1,x,x ** i,r'Plotter instance with $\alpha$',label=[str(i)])
    b.plot(1,x,(-x) ** i,r'Plotter instance with $\beta$',label=[str(i)
       ])
```

Listing 7.14: Code Listing in cell 60

```
plots = []
for i,name in enumerate(['a','b','c']):
    plots.append(ryplot.Plotter(i,1,1,figsize=(5,5)))
    p = plots[-1]
    p.plot(1, x,x ** (i+3))
```

Listing 7.15: Code Listing in cell 63

```
htmlstr = '<p><img src="images/flow2-zzz.png" width=200 />'
HTML(htmlstr)
```

Listing 7.16: Code Listing in cell 72

```
# type(eval("this is a test string")) is not list
```

Listing 7.17: Code Listing in cell 88

```
import pandas as pd
d = {'one' : pd.Series([1., 2., 3.], index=['a', 'b', 'c']),
     'two' : pd.Series([1., 2., 3., 4.], index=['a', 'b', 'c', 'd'])}

df = pd.DataFrame(d)

HTML(df.to_html())
```

Listing 7.18: Code Listing in cell 93

```
# to create the dataframe and write the output to LaTeX object at
   default font size
import pandas as pd
import numpy as np
from IPython.display import Latex

def makeDateRand(nrows=5, ncols=4, cols=list('ABCD')):
    dates = pd.date_range('20130101',periods=nrows)
    df = pd.DataFrame(np.random.randn(nrows,ncols),index=dates,columns=
       cols)
    return df

df = makeDateRand(nrows=3, ncols=5,cols=list('ABCDE'))
lstr = df.to_latex(column_format='|l|l|l|p{50mm}|c|r|')
Latex(lstr)
```

Listing 7.19: Code Listing in cell 94

```
lstrno = df.to_latex(index=False,column_format='|l|l|p{50mm}|c|r|')
Latex(lstrno)
```

Listing 7.20: Code Listing in cell 96

```
# to write the table in LARGE

Latex(lstr)
```

Listing 7.21: Code Listing in cell 98

```
# to write a floating table in tiny font size

Latex(lstr)
```

Listing 7.22: Code Listing in cell 102

```
# display a tikz graphic
lstr = r"""
    \begin{tikzpicture}[scale=4.5, line join=bevel]
      % \a and \b are two macros defining characteristic
      % dimensions of the impossible brick.
      \pgfmathsetmacro{\a}{0.18}
      \pgfmathsetmacro{\b}{1.37}

      \tikzset{%
        apply style/.code={\tikzset{#1}},
        brick_edges/.style={thick,draw=black},
        face_colourA/.style={fill=gray!50},
        face_colourB/.style={fill=gray!25},
        face_colourC/.style={fill=gray!90},
      }

      \foreach \theta/\v/\facestyleone/\facestyletwo in {%
        0/0/{brick_edges,face_colourA}/{brick_edges,face_colourC},
        180/-\a/{brick_edges,face_colourB}/{brick_edges,face_colourC}
      }{
      \begin{scope}[rotate=\theta,shift={(\v,0)}]
        \draw[apply style/.expand once=\facestyleone]
          ({-.5*\b},{1.5*\a}) --
          ++(\b,0)            --
          ++(-\a,-\a)         --
          ++({-\b+2*\a},0)    --
          ++(0,-{2*\a})       --
          ++(\b,0)            --
          ++(-\a,-\a)         --
          ++(-\b,0)           --
          cycle;
        \draw[apply style/.expand once=\facestyletwo]
          ({.5*\b},{1.5*\a})  --
          ++(0,{-2*\a})       --
          ++(-\a,0)           --
          ++(0,\a)            --
          cycle;
        \end{scope}
      }
    \end{tikzpicture}
"""

Latex(lstr)
```

Listing 7.23: Code Listing in cell 103

```
# display another tikz graphic, as a floating figure
```

```
lstr = r"""
\begin{tikzpicture}[font = \sansmath]
  \coordinate (O) at (0,0);

  % ball background color
  \shade[ball color = blue, opacity = 0.2] (0,0) circle [radius = 2cm];

  % cone
  \begin{scope}
    \def\rx{0.71}% horizontal radius of the ellipse
    \def\ry{0.15}% vertical radius of the ellipse
    \def\z{0.725}% distance from center of ellipse to origin

    \path [name path = ellipse]    (0,\z) ellipse ({\rx} and {\ry});
    \path [name path = horizontal] (-\rx,\z-\ry*\ry/\z)
                                   -- (\rx,\z-\ry*\ry/\z);
    \path [name intersections = {of = ellipse and horizontal}];

    % radius to base of cone in ball
    \draw[fill = gray!50, gray!50] (intersection-1) -- (0,0)
      -- (intersection-2) -- cycle;
    % base of cone in ball
    \draw[fill = gray!30, densely dashed] (0,\z) ellipse ({\rx} and {\↵
       ry});
  \end{scope}

  % label of cone
  \draw (0.25,0.4) -- (0.9,0.1) node at (1.05,0.0) {$q$};

  % ball
  \draw (O) circle [radius=2cm];
  % label of ball center point
  \filldraw (O) circle (1pt) node[below] {$P$};

  % radius
  \draw[densely dashed] (O) to [edge label = $r$] (-1.33,1.33);
  \draw[densely dashed] (O) -- (1.33,1.33);

  % cut of ball surface
  \draw[red] (-1.35,1.47) arc [start angle = 140, end angle = 40,
    x radius = 17.6mm, y radius = 14.75mm];
  \draw[red, densely dashed] (-1.36,1.46) arc [start angle = 170, end ↵
     angle = 10,
    x radius = 13.8mm, y radius = 3.6mm];
  \draw[red] (-1.29,1.52) arc [start angle=-200, end angle = 20,
    x radius = 13.75mm, y radius = 3.15mm];

  % label of cut of ball surface
  \draw (-1.2,2.2) -- (-0.53,1.83) node at (-1.37,2.37) {$A$};
\end{tikzpicture}
"""

Latex(lstr)
```

Listing 7.24: Code Listing in cell 105

```
##
lstr = '\n\nThis paragraph is embedded as \LaTeX{} in a Jupyter output ↵
   cell. '\
```

```
            + 'One day this could be a complex computed \LaTeX{} string↵
               .  '\
            + 'This example only has rising rules...  \n\n'
for height in range(1,20):
    lstr += '\\rule[0pt]{{10pt}}{}{}pt{} '.format('{',height,'}')
Latex(lstr + '\n')
```

Listing 7.25: Code Listing in cell 107

```
# to format a numy array in \LaTeX{}
import numpy as np
# install with pip install pylatex
import pylatex as pyl

arr = makeDateRand(nrows=6, ncols=4).values
matrix = pyl.Matrix(arr, mtype='b')
math = pyl.Math(data=['arr=', matrix])
Latex(math.dumps_as_content())
```

Listing 7.26: Code Listing in cell 127

```
##
# to get software versions
# https://github.com/rasbt/watermark
# you only need to do this once
# pip install watermark

%load_ext watermark
%watermark -v -m -p numpy,scipy,pyradi -g
```

BIBLIOGRAPHY

[1] [Online]. Available: http://ipython.org/

[2] [Online]. Available: https://www.youtube.com/watch?v=aIXED26Wppg

[3] J. M. Wing, “Computational thinking,” *Communications of the ACM*, vol. 49, no. 3, pp. 33–35, 2006.

[4] G. Cathedral, “The cathedral labyrinths.” [Online]. Available: http://www.gracecathedral.org/labyrinth/

[5] [Online]. Available: http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.to_latex.html

[6] [Online]. Available: https://github.com/JelteF/PyLaTeX/

[7] [Online]. Available: https://jeltef.github.io/PyLaTeX/current/usage.html#the-classes

[8] [Online]. Available: https://en.wikipedia.org/wiki/Wrapped_normal_distribution