Misfortune

Release 0.0.0

Aug 29, 2020

CONTENTS

CHAPTER

ONE

MISFORTUNE

Misfortune is an idea of converting Markdown documents to Latexpdf using Mistune and Pylatex.

The file `example-1.md`:

```
# Header One

## Subheader One

Text one.

## Subheader Two

Text two.
```

Some code for version 0.0.0:

```
import mistune
import pprint
from pylatex import (
  Document, Section, Subsection, Command)
from pylatex.utils import italic, NoEscape


markdown = mistune.create_markdown (
  renderer=mistune.AstRenderer ())

f = open ('example-1.md')
content = f.read ()
f.close ()

""" example-5.md:
# Header One

## Subheader One

Text one.

## Subheader Two

Text two.
"""

pp = pprint.PrettyPrinter (width=65)
pp.pprint (content)

""" prints:
('# Header One\n'
 '\n'
 '## Subheader One\n'
```

(continues on next page)

(continued from previous page)

```
  '\n'
  'Text one.\n'
  '\n'
  '## Subheader Two\n'
  '\n'
  'Text two.\n'
  '\n')
"""

md = markdown (content)

pp.pprint (md)

""" prints:
[{'children': [{'text': 'Header One', 'type': 'text'}],
   'level': 1,
   'type': 'heading'},
  {'children': [{'text': 'Subheader One', 'type': 'text
↪'}],
   'level': 2,
   'type': 'heading'},
  {'children': [{'text': 'Text one.', 'type': 'text'}],
   'type': 'paragraph'},
  {'children': [{'text': 'Subheader Two', 'type': 'text
↪'}],
   'level': 2,
   'type': 'heading'},
  {'children': [{'text': 'Text two.', 'type': 'text'}],
   'type': 'paragraph'}]
"""

def node2 (m):
   result = ""
   if m ['type'] == 'text':
```

(continues on next page)

(continued from previous page)

```
    result = m ['text']
  return result

def loop2 (children):
  result = ""
  for m in children:
    result += node2 (m)
  return result

def node1 (m):
  result = ""
  if m ['type'] == 'heading':
    if m ['level'] == 1:
      result = Section (loop2 (m ['children']))
    if m ['level'] == 2:
      result = Subsection (loop2 (m ['children']))
  if m ['type'] == 'paragraph':
    result = loop2 (m ['children'])
  return result

doc = Document ("example-1")


for m in md:
  doc.append (node1 (m))

doc.generate_pdf (clean_tex=False)
```

What follows, is a collection of documentation from Mistune and Pylatex. It has been produced using Sphinx without an understanding of how Sphinx works.

CHAPTER

TWO

HOW TO USE MISTUNE

Mistune is easy to use. Here is how you can convert Markdown formatted text into HTML:

```
import mistune

mistune.html(YOUR_MARKDOWN_TEXT)
```

The `.html()` methods has enabled all the features you might want by default:

- No escape of HTML tags
- With **strikethrough** plugin
- With **table** plugin
- With **footnote** plugin

2.1 Customize Mistune

Mistune provides a function to create Markdown instance easily:

```
import mistune

markdown = mistune.create_markdown()
```

This method will create a "escaped" Markdown instance without any plugins, which means:

```
> markdown('<div>hello</div>')
'<p>&lt;div&gt;hello&lt;/div&gt;</p>'
```

Non escaped version:

```
> markdown = mistune.create_markdown(escape=False)
> markdown('<div>hello</div>')
'<div>hello</div>'
```

Adding plugins:

```
> markdown = mistune.create_markdown()
> markdown('~~s~~')
'<p>~~s~~</p>'

> markdown = mistune.create_markdown(
    plugins=['strikethrough'])
> markdown('~~s~~')
'<p><del>s</del></p>'
```

Find out what plugins mistune has built-in in *Mistune Markdown Plugins* sections.

2.2 Customize Renderer

Mistune supports renderer feature which enables developers to customize the output. For instance, to add code syntax highlight:

```
import mistune
from pygments import highlight
from pygments.lexers import get_lexer_by_name
from pygments.formatters import html
```

(continues on next page)

(continued from previous page)

```
class HighlightRenderer(mistune.HTMLRenderer):
  def block_code(self, code, lang=None):
    if lang:
      lexer = get_lexer_by_name(lang, stripall=True)
      formatter = html.HtmlFormatter()
      return highlight(code, lexer, formatter)
    return '<pre><code>' + mistune.escape(code) +
      '</code></pre>'

> markdown = mistune.create_markdown(
    renderer=HighlightRenderer())
> print(markdown('```python\nassert 1 == 1\n```'))

<div class="highlight"><pre>
<span></span>
<span class="k">assert</span>
<span class="mi">1</span>
<span class="o">==</span>
<span class="mi">1</span>
</pre></div>
```

In this way, we can use Pygments to highlight the fenced code. Learn more at *Use renderers*.

2.3 AstRenderer

Mistune can produce AST by default with `mistune.AstRenderer`:

```
markdown = mistune.create_markdown (
  renderer=mistune.AstRenderer())
```

The attributes of `mistune.AstRenderer`:

```
> dir (mistune.AstRenderer())

...
'block_code', 'block_html', 'codespan', 'heading',
'image', 'inline_html', 'linebreak', 'link', 'list',
'list_item', 'newline', 'register', 'text',
'thematic_break']
```

File `example-5.md`:

```
# Header One

## Subheader One

Text one.

## Subheader Two

Text two.
```

This `markdown` function will generate tokens instead of HTML:

```
> markdown = mistune.create_markdown (
    renderer=mistune.AstRenderer())
> f = open ('example-5.md')
> content = f.read ()

> content
('# Header One\n'
 '\n'
 '## Subheader One\n'
 '\n'
 'Text one.\n'
 '\n'
 '## Subheader Two\n'
```

(continues on next page)

(continued from previous page)

```
 '\n'
 'Text two.\n'
 '\n')

> markdown (content)

[ { 'children': [{'text': 'Header One',
                  'type': 'text'}],
    'level': 1,
    'type': 'heading'},

  { 'children': [ { 'text': 'Subheader One',
                    'type': 'text'}],
    'level': 2,
    'type': 'heading'},

  { 'children': [{'text': 'Text one.',
                  'type': 'text'}],
    'type': 'paragraph'},

  { 'children': [ { 'text': 'Subheader Two',
                    'type': 'text'}],
    'level': 2,
    'type': 'heading'},

  { 'children': [{'text': 'Text two.',
                  'type': 'text'}],
    'type': 'paragraph'} ]
```

CHAPTER

THREE

MISTUNE MARKDOWN PLUGINS

3.1 Strikethrough

```
~~here is the content~~
```

`mistune.html()` has enabled strikethrough plugin by default. To create a markdown instance your own:

```
markdown = mistune.create_markdown(
  plugins=['strikethrough'])
```

Another way to create your own Markdown instance:

```
from mistune.plugins import plugin_strikethrough

renderer = mistune.HTMLRenderer()
markdown = mistune.Markdown(renderer,
  plugins=[plugin_strikethrough])
```

3.2 Footnotes

```
content in paragraph with footnote[^1] markup.

[^1]: footnote explain
```

`mistune.html()` has enabled footnote plugin by default. To create a markdown instance your own:

```
markdown = mistune.create_markdown(
  plugins=['footnotes'])
```

Another way to create your own Markdown instance:

```
from mistune.plugins import plugin_footnotes

renderer = mistune.HTMLRenderer()
markdown = mistune.Markdown(renderer,
  plugins=[plugin_footnotes])
```

3.3 Table

Simple formatted table:

```
First Header  | Second Header
------------- | -------------
Content Cell  | Content Cell
Content Cell  | Content Cell
```

Complex formatted table:

```
| First Header  | Second Header |
| ------------- | ------------- |
```

(continues on next page)

(continued from previous page)

```
| Content Cell  | Content Cell  |
| Content Cell  | Content Cell  |
```

Align formatted table:

```
  Left Header |  Center Header  | Right Header
:------------ | :-------------: | ------------:
 Content Cell |  Content Cell   | Content Cell


|   Left Header |  Center Header  | Right Header  |
| :------------ | :-------------: | ------------: |
|  Content Cell |  Content Cell   | Content Cell  |
```

`mistune.html()` has enabled table plugin by default. To create a markdown instance your own:

```
markdown = mistune.create_markdown(
  plugins=['table'])
```

Another way to create your own Markdown instance:

```
from mistune.plugins import plugin_table

renderer = mistune.HTMLRenderer()
markdown = mistune.Markdown(renderer,
  plugins=[plugin_table])
```

3.4 URL

URL plugin enables creating link with raw URL by default:

```
For instance, https://typlog.com/
```

Will be converted into:

```
<p>For instance, <a href="https://typlog.com/>https://
↪typlog.com/</a></p>
```

This plugin is **NOT ENABLED** by default in `mistune.html()`. Mistune values explicit, and we suggest writers to write links in:

```
<https://typlog.com/>
```

To enable **url** plugin with your own markdown instance:

```
markdown = mistune.create_markdown(
  plugins=['url'])
```

Another way to create your own Markdown instance:

```
from mistune.plugins import plugin_url

renderer = mistune.HTMLRenderer()
markdown = mistune.Markdown(renderer,
  plugins=[plugin_url])
```

3.5 Task lists

Task lists plugin enables creating GitHub todo items:

```
- [x] item 1
- [ ] item 2
```

Will be converted into:

```
<ul>
<li class="task-list-item">
 <input class="task-list-item-checkbox"
  type="checkbox" disabled checked/>item 1</li>
<li class="task-list-item">
 <input class="task-list-item-checkbox"
  type="checkbox" disabled/>item 2</li>
</ul>
```

This plugin is **NOT ENABLED** by default in `mistune.html()`. To enable **task_lists** plugin with your own markdown instance:

```
markdown = mistune.create_markdown(
  plugins=['task_lists'])
```

Another way to create your own Markdown instance:

```
from mistune.plugins import plugin_task_lists

renderer = mistune.HTMLRenderer()
markdown = mistune.Markdown(renderer,
  plugins=[plugin_task_lists])
```

CHAPTER

FOUR

DIRECTIVES

Directive is a special plugin which is inspired by reStructuredText. The syntax is very powerful:

```
.. directive-name:: directive value
   :option-key: option value
   :option-key: option value

   full featured markdown text here
```

It was designed to be used by other plugins. There are three built-in plugins based on directive.

4.1 Admonitions

```
.. warning::

   You are looking at the **dev** documentation. Check
   out our [stable](/stable/) documentation instead.
```

Admonitions contains a group of `directive-name`:

```
attention  caution  danger  error
hint  important  note  tip  warning
```

To enable admonitions:

```
import mistune
from mistune.directives import Admonition

markdown = mistune.create_markdown(
    plugins=[Admonition()]
)
```

4.2 TOC Plugin

```
.. toc:: Table of Contents
   :depth: 3
```

TOC plugin is based on directive. It can add a table of contents section in the documentation. Let's take an example:

```
Here is the first paragraph, and we put TOC below.

.. toc::

# H1 title

## H2 title

# H1 title
```

The rendered HTML will show a TOC at the `.. toc::` position. To enable TOC plugin:

```
import mistune
from mistune.directives import DirectiveToc

markdown = mistune.create_markdown(
    plugins=[DirectiveToc()]
)
```

4.3 Include

```
.. include:: hello.md
```

include is a powerful plugin for documentation generator. With this plugin, we can embed contents from other files.

CHAPTER

FIVE

ADVANCED GUIDE OF MISTUNE

5.1 Use renderers

You can customize HTML output with your own renderers. Create a subclass of `mistune.HTMLRenderer`:

```
class MyRenderer(mistune.HTMLRenderer):
  def codespan(self, text):
    if text.startswith('$') and text.endswith('$'):
      return '<span class="math">' + escape(text) +
        '</span>'
    return '<code>' + escape(text) + '</code>'

# use customized renderer
markdown = mistune.create_markdown(
  renderer=MyRenderer())
print(markdown('hi `$a^2=4$`'))
```

Here is a a list of available renderer functions:

```
# inline level
text(self, text)
link(self, link, text=None, title=None)
image(self, src, alt="", title=None)
```

(continues on next page)

(continued from previous page)

```
emphasis(self, text)
strong(self, text)
codespan(self, text)
linebreak(self)
newline(self)
inline_html(self, html)

# block level
paragraph(self, text)
heading(self, text, level)
thematic_break(self)
block_text(self, text)
block_code(self, code, info=None)
block_quote(self, text)
block_html(self, html)
block_error(self, html)
list(self, text, ordered, level, start=None)
list_item(self, text, level)

# provided by strikethrough plugin
strikethrough(self, text)

# provided by table plugin
table(self, text)
table_head(self, text)
table_body(self, text)
table_row(self, text)
table_cell(self, text, align=None, is_head=False)

# provided by footnotes plugin
footnote_ref(self, key, index)
footnotes(self, text)
footnote_item(self, text, key, index)
```

5.2 Create plugins

Mistune has some built-in plugins, you can take a look at the source code in `mistune/plugins` to find out how to write a plugin. Let's take an example for GitHub Wiki links: `[[Page 2|Page 2]]`.

A mistune plugin usually looks like:

```
# define regex for Wiki links
WIKI_PATTERN = (
    r'\[\['                    # [[
    r'([\s\S]+?\|[\s\S]+?)'    # Page 2|Page 2
    r'\]\](?!\])'              # ]]
)

# define how to parse matched item
def parse_wiki(self, m, state):
  # ``self`` is ``md.inline``, see below
  # ``m`` is matched regex item
  text = m.group(1)
  title, link = text.split('|')
  return 'wiki', link, title

# define how to render HTML
def render_html_wiki(link, title):
  return f'<a href="{link}">{title}</a>'

def plugin_wiki(md):
  # this is an inline grammar, so we register wiki
  # rule into md.inline
  md.inline.register_rule(
    'wiki', WIKI_PATTERN, parse_wiki)

  # add wiki rule into active rules
  md.inline.rules.append('wiki')
```

(continues on next page)

(continued from previous page)

```
  # add HTML renderer
  if md.renderer.NAME == 'html':
    md.renderer.register('wiki', render_html_wiki)

# use this plugin
markdown = mistune.create_markdown(
  plugins=[plugin_wiki])
```

Get more examples in `mistune/plugins`, for example, Highlight and Math.

5.3 Highlight

The file highlight.py:

```
import mistune
from pygments import highlight
from pygments.lexers import get_lexer_by_name
from pygments.formatters import HtmlFormatter


def block_code (
  text, lang, inlinestyles=False, linenos=False):
 if not lang:
  text = text.strip()
  return u'<pre><code>%s</code></pre>\n'
    % mistune.escape(text)

 try:
  lexer = get_lexer_by_name(lang, stripall=True)
  formatter = HtmlFormatter(
      noclasses=inlinestyles, linenos=linenos
  )
```

(continues on next page)

(continued from previous page)

```
  code = highlight(text, lexer, formatter)
  if linenos:
    return '<div class="highlight-wrapper">%s</div>\n'
      % code
  return code
 except:
  return '<pre class="%s"><code>%s</code></pre>\n' % (
      lang, mistune.escape(text)
  )


class HighlightMixin(object):
 def block_code(self, text, lang):
  # renderer has an options
  inlinestyles = self.options.get(
    'inlinestyles', False)
  linenos = self.options.get('linenos', False)
  return block_code(text, lang, inlinestyles, linenos
```

5.4 Math

The file math.py:

```
import re

class MathBlockMixin(object):
 """Math mixin for BlockLexer, mix this with
 BlockLexer::

  class MathBlockLexer(MathBlockMixin, BlockLexer):
   def __init__(self, *args, **kwargs):
    super (MathBlockLexer, self).__init__ (
```

(continues on next page)

(continued from previous page)

```
        *args, **kwargs)
    self.enable_math()
"""
def enable_math(self):
 self.rules.block_math = re.compile(
   r'^\$\$(.*?)\$\$', re.DOTALL)
 self.rules.block_latex = re.compile(
   r'^\\begin\{([a-z]*\*?)\}(.*?)\\end\{\1\}',
   re.DOTALL
 )
 self.default_rules.extend([
   'block_math', 'block_latex'])

def parse_block_math(self, m):
 """Parse a $$math$$ block"""
 self.tokens.append({
  'type': 'block_math',
  'text': m.group(1)
 })

def parse_block_latex(self, m):
 self.tokens.append({
  'type': 'block_latex',
  'name': m.group(1),
  'text': m.group(2)
 })


class MathInlineMixin(object):
 """Math mixin for InlineLexer, mix this with
 InlineLexer::

  class MathInlineLexer(InlineLexer, MathInlineMixin):
   def __init__(self, *args, **kwargs):
```

(continues on next page)

(continued from previous page)

```
    super (
      MathInlineLexer, self).__init__(*args, **kwargs)
    self.enable_math()
 """

 def enable_math(self):
  self.rules.math = re.compile(r'^\$(.+?)\$')
  self.default_rules.insert(0, 'math')
  self.rules.text = re.compile(
   r'^[\s\S]+?(?=[\\<!\[_*`~\$]|https?://| {2,}\n|$)')

 def output_math(self, m):
  return self.renderer.math(m.group(1))


class MathRendererMixin(object):
 def block_math(self, text):
  return '$$%s$$' % text

 def block_latex(self, name, text):
  return r'\begin{%s}%s\end{%s}' % (name, text, name)

 def math(self, text):
  return '$%s$' % text
```

5.5 Write directives

CHAPTER

SIX

API REFERENCE

mistune.**html**(*text*)

Parameters **text** – markdown formatted text

Turn markdown text into HTML without escaping. For instance:

```
> text = '**hello** <span>world</span>'
> mistune.html(text)
'<p><strong>hello</strong> <span>world</span></p>'
```

mistune.**create_markdown**(*escape=True*, *renderer=None*, *plugins=None*)

Create a Markdown instance based on the given condition.

Parameters

- **escape** – Boolean. If using html renderer, escape html.
- **renderer** – renderer instance or string of `html` and `ast`.
- **plugins** – List of plugins, string or callable.

This method is used when you want to re-use a Markdown instance:

```
markdown = create_markdown(
    escape=False,
```

(continues on next page)

(continued from previous page)

```
    renderer='html',
    plugins=['url', 'strikethrough', 'footnotes',
↪'table'],
)
# re-use markdown function
markdown('.... your text ...')
```

CHAPTER

SEVEN

DEVELOPER GUIDE

Here is the API reference for mistune.

class mistune.**Renderer**(***kwargs*)

The default HTML renderer for rendering Markdown.

autolink(*link*, *is_email=False*)

Rendering a given link or email address.

Parameters:

- **link** – link content or email address.
- **is_email** – whether this is an email or not.

block_code(*code*, *lang=None*)

Rendering block level code. “pre > code”.

Parameters:

- **code** – text content of the code block.
- **lang** – language of the given code.

block_html(*html*)

Rendering block level pure html content.

Parameters: **html** – text content of the html snippet.

block_quote(*text*)

Rendering <blockquote> with the given text.

Parameters: **text** – text content of the blockquote.

codespan(*text*)

Rendering inline *code* text.

Parameters: **text** – text content for inline code.

double_emphasis(*text*)
Rendering **strong** text.
Parameters: **text** – text content for emphasis.

emphasis(*text*)
Rendering *emphasis* text.
Parameters: **text** – text content for emphasis.

escape(*text*)
Rendering escape sequence.
Parameters: **text** – text content.

footnote_item(*key*, *text*)
Rendering a footnote item.
Parameters:
- **key** – identity key for the footnote.
- **text** – text content of the footnote.

footnote_ref(*key*, *index*)
Rendering the ref anchor of a footnote.
Parameters:
- **key** – identity key for the footnote.
- **index** – the index count of current footnote.

footnotes(*text*)
Wrapper for all footnotes.
Parameters: **text** – contents of all footnotes.

header(*text*, *level*, *raw=None*)
Rendering header/heading tags like “<h1>” “<h2>”.
Parameters:
- **text** – rendered text content for the header.
- **level** – a number for the header level, for example: 1.
- **raw** – raw text content of the header.

hrule()
Rendering method for “<hr>” tag.

image(*src*, *title*, *text*)
Rendering a image with title and text.
Parameters:
- **src** – source link of the image.
- **title** – title text of the image.
- **text** – alt text of the image.

inline_html(*html*)
Rendering span level pure html content.
Parameters: **html** – text content of the html snippet.

linebreak()
Rendering line break like "<br>".

link(*link*, *title*, *text*)
Rendering a given link with content and title.
Parameters:
- **link** – href link for "<a>" tag.
- **title** – title content for *title* attribute.
- **text** – text content for description.

list(*body*, *ordered=True*)
Rendering list tags like "<ul>" and "<ol>".
Parameters:
- **body** – body contents of the list.
- **ordered** – whether this list is ordered or not.

list_item(*text*)
Rendering list item snippet. Like "<li>".

newline()
Rendering newline element.

paragraph(*text*)
Rendering paragraph tags. Like "<p>".

placeholder()
Returns the default, empty output value for the renderer.

All renderer methods use the '+=' operator to append to this

value. Default is a string so rendering HTML can build up a result string with the rendered Markdown.

Can be overridden by Renderer subclasses to be types like an empty list, allowing the renderer to create a tree-like structure to represent the document (which can then be reprocessed later into a separate format like docx or pdf).

strikethrough(*text*)
Rendering ~~strikethrough~~ text.
Parameters: **text** – text content for strikethrough.

table(*header*, *body*)
Rendering table element. Wrap header and body in it.
Parameters:
- **header** – header part of the table.
- **body** – body part of the table.

table_cell(*content*, ***flags*)
Rendering a table cell. Like "<th>" "<td>".
Parameters:
- **content** – content of current table cell.
- **header** – whether this is header or not.
- **align** – align of current table cell.

table_row(*content*)
Rendering a table row. Like "<tr>".
Parameters: **content** – content of current table row.

text(*text*)
Rendering unformatted text.
Parameters: **text** – text content.

class mistune.**Markdown**(*renderer=None*, *inline=None*, *block=None*, ***kwargs*)
The Markdown parser.

Parameters:

- **renderer** – An instance of "Renderer".

- **inline** – An inline lexer class or instance.
- **block** – A block lexer class or instance.

render(*text*)

Render the Markdown text.

Parameters: **text** – markdown formatted text content.

mistune.**markdown**(*text*, *escape=True*, ***kwargs*)

Render markdown formatted text to html.

Parameters:

- **text** – markdown formatted text content.
- **escape** – if set to False, all html tags will not be escaped.
- **use_xhtml** – output with xhtml tags.
- **hard_wrap** – if set to True, it will use the GFM line breaks feature.
- **parse_block_html** – parse text only in block level html.
- **parse_inline_html** – parse text only in inline level html.

mistune.**escape**(*text*, *quote=False*, *smart_amp=True*)

Replace special characters "&", "<" and ">" to HTML-safe sequences.

The original cgi.escape will always escape "&", but you can control this one for a smart escape amp.

Parameters:

- **quote** – if set to True, " and ' will be escaped.
- **smart_amp** – if set to False, & will always be escaped.

CHAPTER

EIGHT

PYLATEX EXAMPLES

Module pylatex:

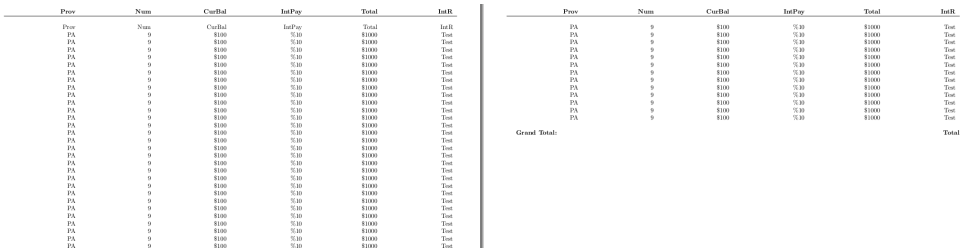
```
> import pylatex
> dir (pylatex)
['Alignat', 'Autoref', 'Axis', 'Center', 'ColumnType',
 'Command', 'Description', 'Document', 'Enumerate',
 'Eqref', 'FBox', 'Figure', 'FlushLeft', 'FlushRight',
 'Foot', 'FootnoteText', 'HFill', 'Head',
 'HorizontalSpace', 'HugeText', 'Hyperref', 'Itemize',
 'Label', 'LargeText', 'LineBreak', 'LongTable',
 'LongTabu', 'LongTabularx', 'Marker', 'Math',
 'Matrix', 'MdFramed', 'MediumText', 'MiniPage',
 'MultiColumn', 'MultiRow', 'NewLine', 'NewPage',
 'NoEscape', 'Package', 'PageStyle', 'Pageref', 'Plot',
 'Quantity', 'Ref', 'Section', 'SmallText',
 'StandAloneGraphic', 'SubFigure', 'Subsection',
 'Subsubsection', 'Table', 'TableRowSizeError', 'Tabu',
 'Tabular', 'Tabularx', 'TextBlock', 'TextColor',
 'TikZ', 'TikZCoordinate', 'TikZDraw', 'TikZNode',
 'TikZNodeAnchor', 'TikZOptions', 'TikZPath',
 'TikZPathList', 'TikZScope', 'TikZUserPath',
 'UnsafeCommand', 'VectorName', 'VerticalSpace',
...
```

(continues on next page)

(continued from previous page)

```
 'base_classes', 'basic', 'config', 'document',
 'errors', 'escape_latex', 'figure', 'frames',
 'headfoot', 'labelref', 'lists', 'math', 'package',
 'position', 'quantities', 'section',
 'simple_page_number', 'table', 'tikz', 'utils']

> dir (pylatex.document)
['Command', 'CompilerError', 'Container', 'Document',
 'Environment', 'LatexObject', 'NoEscape', 'Package',
 'UnsafeCommand', ...]

> dir (pylatex.Document)
[..., 'add_color', 'append', 'begin_paragraph',
 'change_document_style', 'change_length',
 'change_page_style', 'clear', 'content_separator',
 'copy', 'count', 'create', 'dump', 'dump_packages',
 'dumps', 'dumps_as_content', 'dumps_content',
 'dumps_packages', 'end_paragraph', 'escape',
 'extend', 'generate_pdf', 'generate_tex', 'index',
 'insert', 'latex_name', 'omit_if_empty', 'packages',
 'pop', 'remove', 'reverse', 'separate_paragraph',
 'set_variable', 'sort']

> dir (pylatex.utils)
['NoEscape', ..., 'bold', 'dumps_list', 'escape_latex',
 'fix_filename', 'italic', 'make_temp_dir', 'os',
 'pylatex', 'rm_temp_dir', 'shutil', 'tempfile',
 'verbatim']
```

8.1 LongTabu



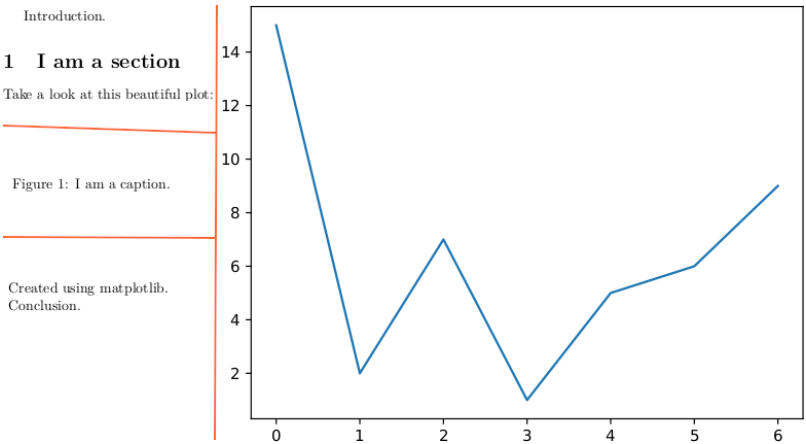
Prov	Num	CurBal	IntPay	Total	IntR
Prov	Num	CurBal	IntPay	Total	IntR
PA	9	$100	%10	$1000	Test
PA	9	$100	%10	$1000	Test
PA	9	$100	%10	$1000	Test

(continued from previous page)

```
  data_table.add_empty_row()
  data_table.end_table_header()
  data_table.add_row([
   "Prov", "Num", "CurBal", "IntPay", "Total", "IntR"])
  row = ["PA", "9", "$100", "%10", "$1000", "Test"]
  for i in range(50):
      data_table.add_row(row)

 doc.append(bold("Grand Total:"))
 doc.append(HFill())
 doc.append(bold("Total"))

 doc.generate_pdf("longtabu", clean_tex=False)

generate_longtabu()
```

8.2 Matplotlib

This example shows matplotlib functionality.



The file matplotlib_ex.py:

```
import matplotlib
matplotlib.use('Agg')
from pylatex import Document, Section, Figure, NoEscape
import matplotlib.pyplot as plt

def main (fname, width, *args, **kwargs):
  geometry_options = {"right": "2cm", "left": "2cm"}
  doc = Document(fname, geometry_options=geometry_
↪options)
  doc.append('Introduction.')
  with doc.create(Section('I am a section')):
   doc.append('Take a look at this beautiful plot:')
   with doc.create(Figure(position='htbp')) as plot:
    plot.add_plot(
      width=NoEscape(width), *args, **kwargs)
    plot.add_caption('I am a caption.')
   doc.append('Created using matplotlib.')
  doc.append('Conclusion.')
  doc.generate_pdf(clean_tex=False)

if __name__ == '__main__':
  x = [0, 1, 2, 3, 4, 5, 6]
  y = [15, 2, 7, 1, 5, 6, 9]
  plt.plot(x, y)
  main('matplotlib_ex-dpi', r'1\textwidth', dpi=300)
  main('matplotlib_ex-facecolor',
    r'0.5\textwidth', facecolor='b')
```

8.3 Header

This example shows the functionality of the PageHeader object.

It creates a sample page with the different types of headers and footers.

The file header.py:

```
from pylatex import (
  Document, PageStyle, Head, MiniPage, Foot, LargeText,
  MediumText, LineBreak, simple_page_number)
from pylatex.utils import bold


def generate_header():
  geometry_options = {"margin": "0.7in"}
  doc = Document(geometry_options=geometry_options)
  # Add document header
  header = PageStyle("header")
  # Create left header
  with header.create(Head("L")):
      header.append("Page date: ")
      header.append(LineBreak())
      header.append("R3")
  # Create center header
  with header.create(Head("C")):
      header.append("Company")
  # Create right header
  with header.create(Head("R")):
      header.append(simple_page_number())
  # Create left footer
  with header.create(Foot("L")):
      header.append("Left Footer")
  # Create center footer
  with header.create(Foot("C")):
```

(continues on next page)

(continued from previous page)

```
        header.append("Center Footer")
    # Create right footer
    with header.create(Foot("R")):
        header.append("Right Footer")

    doc.preamble.append(header)
    doc.change_document_style("header")

    # Add Heading
    with doc.create(MiniPage(align='c')):
        doc.append(LargeText(bold("Title")))
        doc.append(LineBreak())
        doc.append(MediumText(bold("As at:")))

    doc.generate_pdf("header", clean_tex=False)

generate_header()
```

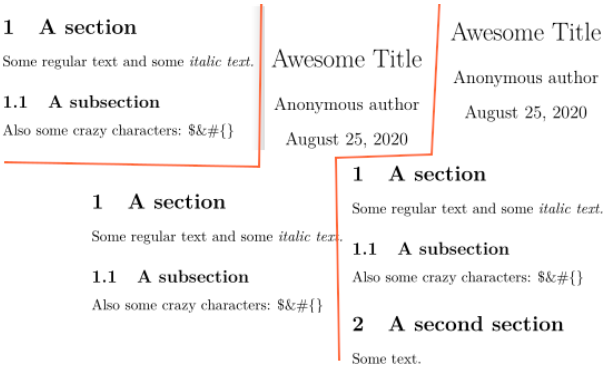
8.4 os-walk-1.py

The file os-walk-1.py:

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

8.5 Basic

This example shows basic document generation functionality.



The file basic.py:

```
from pylatex import (
  Document, Section, Subsection, Command)
from pylatex.utils import italic, NoEscape

def fill_document(doc):
 """Add a section, a subsection and some text to the
    document.
 :param doc: the document
 :type doc: :class:`pylatex.document.Document` instance
 """
 with doc.create(Section('A section')):
  doc.append('Some regular text and some ')
  doc.append(italic('italic text. '))

  with doc.create(Subsection('A subsection')):
   doc.append('Also some crazy characters: $&#{}')

# Basic document
doc = Document('basic')
```

(continues on next page)

(continued from previous page)

```
fill_document(doc)

doc.generate_pdf(clean_tex=False)
doc.generate_tex()

# Document with `\maketitle` command activated
doc = Document()

doc.preamble.append(Command(
  'title', 'Awesome Title'))
doc.preamble.append(Command(
  'author', 'Anonymous author'))
doc.preamble.append(Command(
  'date', NoEscape(r'\today')))
doc.append(NoEscape(r'\maketitle'))

fill_document(doc)

doc.generate_pdf('basic_maketitle', clean_tex=False)

# Add stuff to the document
with doc.create(Section('A second section')):
    doc.append('Some text.')

doc.generate_pdf('basic_maketitle2', clean_tex=False)

# The document as string in LaTeX syntax:
tex = doc.dumps()
```

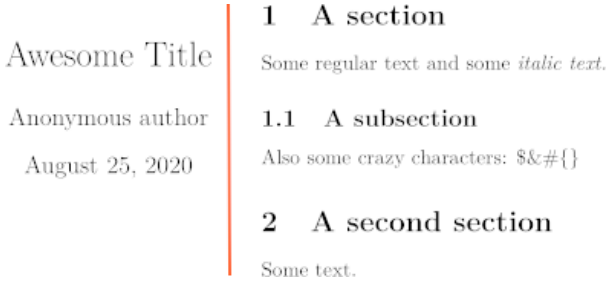
8.6 os-walk-2.py

The file os-walk-2.py:

```
import os

path = r"."

for dirname, subdirs, files in os.walk(path):
  print(dirname)
  print('  subdirs:', subdirs)
  print('  files:', files)
```

8.7 Basic Inheritance

This example shows basic document generation functionality by inheritance.



The file basic_inheritance.py:

```
from pylatex (
  import Document, Section, Subsection, Command)
from pylatex.utils import italic, NoEscape


class MyDocument(Document):
  def __init__(self):
    super().__init__()

    self.preamble.append(Command(
```

(continues on next page)

(continued from previous page)

```
      'title', 'Awesome Title'))
    self.preamble.append(Command(
      'author', 'Anonymous author'))
    self.preamble.append(Command(
      'date', NoEscape(r'\today')))
    self.append(NoEscape(r'\maketitle'))

  def fill_document(self):
    """Add a section, a subsection and some text to
    the document."""
    with self.create(Section('A section')):
     self.append('Some regular text and some ')
     self.append(italic('italic text. '))

     with self.create(Subsection('A subsection')):
      self.append('Also some crazy characters: $&#{}')

# Document
doc = MyDocument()

# Call function to add text
doc.fill_document()

# Add stuff to the document
with doc.create(Section('A second section')):
    doc.append('Some text.')

doc.generate_pdf('basic_inheritance', clean_tex=False)
tex = doc.dumps() # The docum as string in LaTeX syntax
```

8.8 Subfigure

This example shows subfigure functionality.

1 Showing subfigures

(a) Kitten on the left (b) Kitten on the right

Figure 1: Two kittens

The file subfigure.py:

```
from pylatex import
  (Document, Section, Figure, SubFigure, NoEscape)
import os

doc = Document(default_filepath='subfigures')
image_filename = os.path.join(
  os.path.dirname(__file__), 'kitten.jpg')

with doc.create(Section('Showing subfigures')):
 with doc.create(Figure(position='h!')) as kittens:
  with doc.create(SubFigure(
   position='b',
   width=NoEscape(r'0.45\linewidth'))) as left_kitten:

   left_kitten.add_image(image_filename,
                         width=NoEscape(r'\linewidth'))
```

(continues on next page)

(continued from previous page)

```
    left_kitten.add_caption('Kitten on the left')
  with doc.create(SubFigure(
    position='b',
    width=NoEscape(r'0.45\linewidth'))) as right_kitten:

    right_kitten.add_image(
      image_filename,
      width=NoEscape(r'\linewidth'))
    right_kitten.add_caption('Kitten on the right')
  kittens.add_caption("Two kittens")

doc.generate_pdf(clean_tex=False)
```

8.9 Text Block

This example shows the functionality of the TextBlock element.

It creates a sample cheque to demonstrate the positioning of the elements on the page.

The file textblock.py:

```
from pylatex import (
  Document, MiniPage, TextBlock, MediumText, HugeText,
  SmallText, VerticalSpace, HorizontalSpace)
```

(continues on next page)

(continued from previous page)

```
from pylatex.utils import bold

geometry_options = {"margin": "0.5in"}
doc = Document(
  indent=False, geometry_options=geometry_options)
doc.change_length("\TPHorizModule", "1mm")
doc.change_length("\TPVertModule", "1mm")

with doc.create(MiniPage(width=r"\textwidth")) as page:
  with page.create(TextBlock(100, 0, 0)):
   page.append("**** Ten Thousand Dollars")

  with page.create(TextBlock(100, 0, 30)):
   page.append("COMPANY NAME")
   page.append("\nSTREET, ADDRESS")
   page.append("\nCITY, POSTAL CODE")

  with page.create(TextBlock(100, 150, 40)):
   page.append(HugeText(bold("VOID")))

  with page.create(TextBlock(80, 150, 0)):
   page.append("DATE")
   page.append(MediumText(bold("2016 06 07\n")))
   page.append(HorizontalSpace("10mm"))
   page.append(SmallText("Y/A M/M D/J"))

  with page.create(TextBlock(70, 150, 30)):
   page.append(MediumText(bold("$***** 10,000.00")))

  page.append(VerticalSpace("100mm"))

doc.generate_pdf("textblock", clean_tex=False)
```

8.10 Tabus

Prov	Num	CurBal	IntPay	Total	IntR
Prov	Num	CurBal	IntPay	Total	IntR
PA	9	$100	%10	$1000	Test
PA	9	$100	%10	$1000	Test
PA	9	$100	%10	$1000	Test

(continued from previous page)

```
  # Generate data table with 'tight' columns
  fmt = "X[r] X[r] X[r] X[r] X[r] X[r]"
  with doc.create (
            LongTabu(fmt, spread="0pt")) as data_table:
   header_row1 = [
    "Prov", "Num", "CurBal", "IntPay", "Total", "IntR"]
   data_table.add_row(header_row1, mapper=[bold])
   data_table.add_hline()
   data_table.add_empty_row()
   data_table.end_table_header()
   data_table.add_row([
    "Prov", "Num", "CurBal", "IntPay", "Total", "IntR
↪"])
   row = ["PA", "9", "$100", "%10", "$1000", "Test"]
   for i in range(40):
       data_table.add_row(row)

  with doc.create(Center()) as centered:
   with centered.create (
        Tabu("X[r] X[r]", spread="1in")) as data_table:
    header_row1 = ["X", "Y"]
    data_table.add_row(header_row1, mapper=[bold])
    data_table.add_hline()
    row = [randint(0, 1000), randint(0, 1000)]
    for i in range(4):
        data_table.add_row(row)

  with doc.create(Center()) as centered:
   with centered.create (
           Tabu("X[r] X[r]", to="4in")) as data_table:
    header_row1 = ["X", "Y"]
    data_table.add_row(header_row1, mapper=[bold])
    data_table.add_hline()
```

(continues on next page)

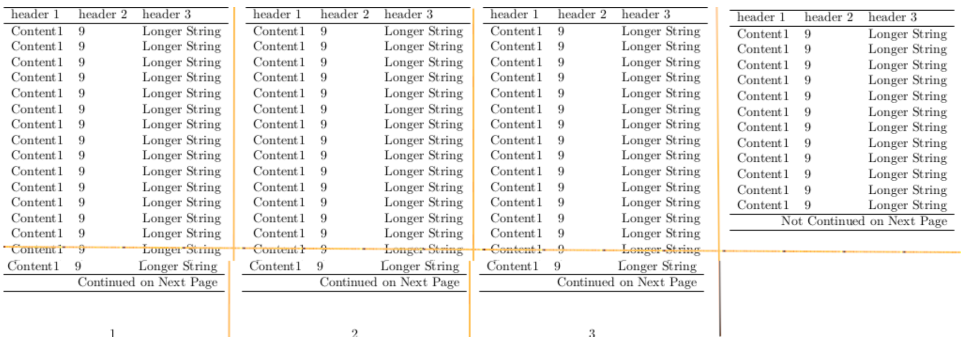
(continued from previous page)

```
    row = [randint(0, 1000), randint(0, 1000)]
    for i in range(4):
        data_table.add_row(row)

  doc.generate_pdf("tabus", clean_tex=False)

generate_tabus()
```

8.11 Long Table

This example shows the functionality of the longtable element.

It creates a sample multi-page spanning table.



header 1	header 2	header 3
Content1	9	Longer String
Content1	9	Longer String
Content1	9	Longer String
Content1	9	Longer String
Content1	9	Longer String
Content1	9	Longer String
Content1	9	Longer String
Content1	9	Longer String
Content1	9	Longer String
Content1	9	Longer String
Content1	9	Longer String
Content1	9	Longer String
Content1	9	Longer String
Content1	9	Longer String
Content1	9	Longer String
Content1	9	Longer String
	Continued on Next Page	

1

header 1	header 2	header 3
Content1	9	Longer String
Content1	9	Longer String
Content1	9	Longer String
Content1	9	Longer String
Content1	9	Longer String
Content1	9	Longer String
Content1	9	Longer String
Content1	9	Longer String
Content1	9	Longer String
Content1	9	Longer String
Content1	9	Longer String
Content1	9	Longer String
Content1	9	Longer String
Content1	9	Longer String
Content1	9	Longer String
Content1	9	Longer String
	Continued on Next Page	

2

header 1	header 2	header 3
Content1	9	Longer String
Content1	9	Longer String
Content1	9	Longer String
Content1	9	Longer String
Content1	9	Longer String
Content1	9	Longer String
Content1	9	Longer String
Content1	9	Longer String
Content1	9	Longer String
Content1	9	Longer String
Content1	9	Longer String
Content1	9	Longer String
Content1	9	Longer String
Content1	9	Longer String
Content1	9	Longer String
Content1	9	Longer String
	Continued on Next Page	

3

header 1	header 2	header 3
Content1	9	Longer String
Content1	9	Longer String
Content1	9	Longer String
Content1	9	Longer String
Content1	9	Longer String
Content1	9	Longer String
Content1	9	Longer String
Content1	9	Longer String
Content1	9	Longer String
Content1	9	Longer String
Content1	9	Longer String
Content1	9	Longer String
	Not Continued on Next Page	

The file longtable.py:

```
from pylatex import Document, LongTable, MultiColumn

def generate_longtabu():
  geometry_options = {
      "margin": "2.54cm",
      "includeheadfoot": True }
  doc = Document(
```

(continues on next page)

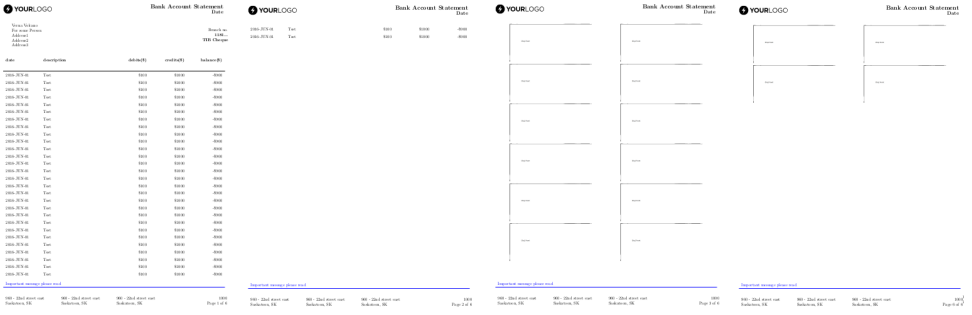
(continued from previous page)

```
    page_numbers=True,
    geometry_options=geometry_options)
  # Generate data table:
  with doc.create(LongTable("l l l")) as data_table:
   data_table.add_hline()
   data_table.add_row([
     "header 1", "header 2", "header 3"])
   data_table.add_hline()
   data_table.end_table_header()
   data_table.add_hline()
   data_table.add_row((
     MultiColumn(3, align='r',
     data='Continued on Next Page'),))
   data_table.add_hline()
   data_table.end_table_footer()
   data_table.add_hline()
   data_table.add_row((
     MultiColumn(3, align='r',
     data='Not Continued on Next Page'),))
   data_table.add_hline()
   data_table.end_table_last_footer()
   row = ["Content1", "9", "Longer String"]
   for i in range(150):
       data_table.add_row(row)
  doc.generate_pdf("longtable", clean_tex=False)
generate_longtabu()
```

8.12 Complex Report

This example shows the functionality of the PyLaTeX library.

It creates a sample report with 2 tables, one containing images and the other containing data. It also creates a complex header with an image.



The file complex_report.py:

```
import os

from pylatex import (
  Document, PageStyle, Head, Foot, MiniPage,
  StandAloneGraphic, MultiColumn, Tabu, LongTabu,
  LargeText, MediumText, LineBreak, NewPage, Tabularx,
  TextColor, simple_page_number)
from pylatex.utils import bold, NoEscape


def generate_unique():
 geometry_options = {
      "head": "40pt",
      "margin": "0.5in",
      "bottom": "0.6in",
      "includeheadfoot": True
 }
 doc = Document(geometry_options=geometry_options)

 # Generating first page style
 first_page = PageStyle("firstpage")

 # Header image
 with first_page.create(Head("L")) as header_left:
```

(continues on next page)

(continued from previous page)

```
    with header_left.create(MiniPage(
      width=NoEscape(r"0.49\textwidth"),
      pos='c')) as logo_wrapper:
     logo_file = os.path.join(
       os.path.dirname(__file__),
       'sample-logo.png')
     logo_wrapper.append(StandAloneGraphic(
       image_options="width=120px",
       filename=logo_file))

# Add document title
with first_page.create(Head("R")) as right_header:
  with right_header.create (MiniPage(
   width=NoEscape(r"0.49\textwidth"),
   pos='c', align='r')) as title_wrapper:
    title_wrapper.append(LargeText(
      bold("Bank Account Statement")))
    title_wrapper.append(LineBreak())
    title_wrapper.append(MediumText(bold("Date")))

# Add footer
with first_page.create(Foot("C")) as footer:
  message = "Important message please read"
  with footer.create(Tabularx(
   "X X X X",
   width_argument=NoEscape(r"\textwidth")
  )) as footer_table:

    footer_table.add_row(
        [MultiColumn(4, align='l',
          data=TextColor("blue", message))])
    footer_table.add_hline(color="blue")
    footer_table.add_empty_row()
```

(continues on next page)

(continued from previous page)

```
    branch_address = MiniPage(
        width=NoEscape(r"0.25\textwidth"),
        pos='t')
    branch_address.append("960 - 22nd street east")
    branch_address.append("\n")
    branch_address.append("Saskatoon, SK")

    document_details = MiniPage(
      width=NoEscape(r"0.25\textwidth"),
      pos='t', align='r')
    document_details.append("1000")
    document_details.append(LineBreak())
    document_details.append(simple_page_number())

    footer_table.add_row([
      branch_address, branch_address,
      branch_address, document_details])

doc.preamble.append(first_page)
# End first page style

# Add customer information
with doc.create(
            Tabu ("X[l] X[r]")) as first_page_table:
 customer = MiniPage (
   width=NoEscape(r"0.49\textwidth"), pos='h')
 customer.append("Verna Volcano")
 customer.append("\n")
 customer.append("For some Person")
 customer.append("\n")
 customer.append("Address1")
 customer.append("\n")
 customer.append("Address2")
 customer.append("\n")
```

(continues on next page)

(continued from previous page)

```
  customer.append("Address3")

  # Add branch information
  branch = MiniPage(
    width=NoEscape(r"0.49\textwidth"),
    pos='t!',
    align='r')
  branch.append("Branch no.")
  branch.append(LineBreak())
  branch.append(bold("1181..."))
  branch.append(LineBreak())
  branch.append(bold("TIB Cheque"))

  first_page_table.add_row([customer, branch])
  first_page_table.add_empty_row()

doc.change_document_style("firstpage")
doc.add_color(
   name="lightgray", model="gray", description="0.80")

# Add statement table
with doc.create(LongTabu(
   "X[l] X[2l] X[r] X[r] X[r]",
   row_height=1.5)) as data_table:
  data_table.add_row([
     "date", "description", "debits($)",
     "credits($)", "balance($)"],
    mapper=bold, color="lightgray")
  data_table.add_empty_row()
  data_table.add_hline()
  row = [
    "2016-JUN-01", "Test", "$100", "$1000", "-$900"]
  for i in range(30):
      if (i % 2) == 0:
```

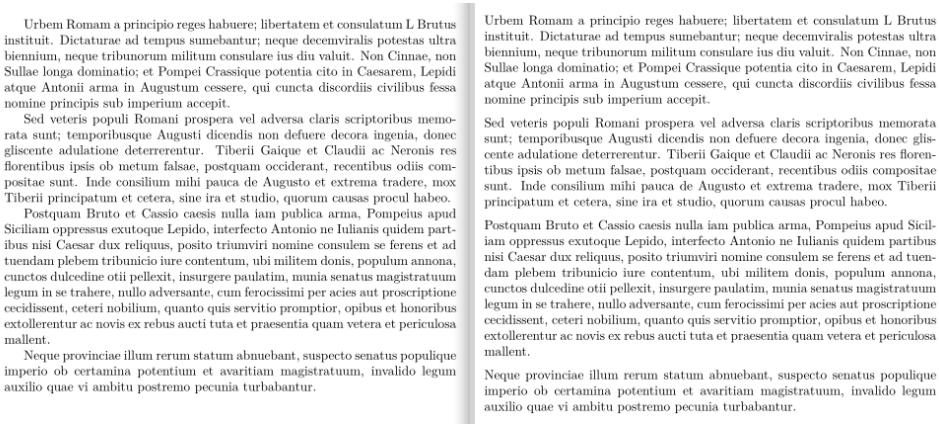
(continues on next page)

(continued from previous page)

```
            data_table.add_row(row, color="lightgray")
        else:
            data_table.add_row(row)
    doc.append(NewPage())

    # Add cheque images
    with doc.create(
                    LongTabu("X[c] X[c]")) as cheque_table:
        cheque_file = os.path.join(
          os.path.dirname(__file__), 'chequeexample.png')
        cheque = StandAloneGraphic(
          cheque_file, image_options="width=200px")
        for i in range(0, 20):
          cheque_table.add_row([cheque, cheque])
    doc.generate_pdf("complex_report", clean_tex=False)
generate_unique()
```

8.13 Configuration

This example shows basic document generation functionality.



Urbem Romam a principio reges habuere; libertatem et consulatum L Brutus instituit. Dictaturae ad tempus sumebantur; neque decemviralis potestas ultra biennium, neque tribunorum militum consulare ius diu valuit. Non Cinnae, non Sullae longa dominatio; et Pompei Crassique potentia cito in Caesarem, Lepidi atque Antonii arma in Augustum cessere, qui cuncta discordiis civilibus fessa nomine principis sub imperium accepit.

Sed veteris populi Romani prospera vel adversa claris scriptoribus memorata sunt; temporibusque Augusti dicendis non defuere decora ingenia, donec gliscente adulatione deterrerentur. Tiberii Gaique et Claudii ac Neronis res florentibus ipsis ob metum falsae, postquam occiderant, recentibus odiis compositae sunt. Inde consilium mihi pauca de Augusto et extrema tradere, mox Tiberii principatum et cetera, sine ira et studio, quorum causas procul habeo.

Postquam Bruto et Cassio caesis nulla iam publica arma, Pompeius apud Siciliam oppressus exutoque Lepido, interfecto Antonio ne Iulianis quidem partibus nisi Caesar dux reliquus, posito triumviri nomine consulem se ferens et ad tuendam plebem tribunicio iure contentum, ubi militem donis, populum annona, cunctos dulcedine otii pellexit, insurgere paulatim, munia senatus magistratuum legum in se trahere, nullo adversante, cum ferocissimi per acies aut proscriptione cecidissent, ceteri nobilium, quanto quis servitio promptior, opibus et honoribus extollerentur ac novis ex rebus aucti tuta et praesentia quam vetera et periculosa mallent.

Neque provinciae illum rerum statum abnuebant, suspecto senatus populique imperio ob certamina potentium et avaritiam magistratuum, invalido legum auxilio quae vi ambitu postremo pecunia turbabantur.

Urbem Romam a principio reges habuere; libertatem et consulatum L Brutus instituit. Dictaturae ad tempus sumebantur; neque decemviralis potestas ultra biennium, neque tribunorum militum consulare ius diu valuit. Non Cinnae, non Sullae longa dominatio; et Pompei Crassique potentia cito in Caesarem, Lepidi atque Antonii arma in Augustum cessere, qui cuncta discordiis civilibus fessa nomine principis sub imperium accepit.

Sed veteris populi Romani prospera vel adversa claris scriptoribus memorata sunt; temporibusque Augusti dicendis non defuere decora ingenia, donec gliscente adulatione deterrerentur. Tiberii Gaique et Claudii ac Neronis res florentibus ipsis ob metum falsae, postquam occiderant, recentibus odiis compositae sunt. Inde consilium mihi pauca de Augusto et extrema tradere, mox Tiberii principatum et cetera, sine ira et studio, quorum causas procul habeo.

Postquam Bruto et Cassio caesis nulla iam publica arma, Pompeius apud Siciliam oppressus exutoque Lepido, interfecto Antonio ne Iulianis quidem partibus nisi Caesar dux reliquus, posito triumviri nomine consulem se ferens et ad tuendam plebem tribunicio iure contentum, ubi militem donis, populum annona, cunctos dulcedine otii pellexit, insurgere paulatim, munia senatus magistratuum legum in se trahere, nullo adversante, cum ferocissimi per acies aut proscriptione cecidissent, ceteri nobilium, quanto quis servitio promptior, opibus et honoribus extollerentur ac novis ex rebus aucti tuta et praesentia quam vetera et periculosa mallent.

Neque provinciae illum rerum statum abnuebant, suspecto senatus populique imperio ob certamina potentium et avaritiam magistratuum, invalido legum auxilio quae vi ambitu postremo pecunia turbabantur.

The file config.py:

```
from pylatex import Document, NoEscape
import pylatex.config as cf

lorem = '''
Urbem Romam a principio reges habuere; libertatem et
consulatum L Brutus instituit. Dictaturae ad tempus
sumebantur; neque decemviralis potestas ultra biennium,
neque tribunorum militum consulare ius diu valuit.
   Non Cinnae, non Sullae longa dominatio; et Pompei
Crassique potentia cito in Caesarem, Lepidi atque
Antonii arma in Augustum cessere, qui cuncta discordiis
civilibus fessa nomine principis sub imperium accepit.
   Sed veteris populi Romani prospera vel adversa
claris scriptoribus memorata sunt; temporibusque
Augusti dicendis non defuere decora ingenia, donec
gliscente adulatione deterrerentur. Tiberii Gaique et
Claudii ac Neronis res florentibus ipsis ob metum
falsae, postquam occiderant, recentibus odiis
compositae sunt. Inde consilium mihi pauca de Augusto
et extrema tradere, mox Tiberii principatum et cetera,
sine ira et studio, quorum causas procul habeo.
   Postquam Bruto et Cassio caesis nulla iam publica
arma, Pompeius apud Siciliam oppressus exutoque Lepido,
interfecto Antonio ne Iulianis quidem partibus nisi
Caesar dux reliquus, posito triumviri nomine consulem
se ferens et ad tuendam plebem tribunicio iure
contentum, ubi militem donis, populum annona, cunctos
dulcedine otii pellexit, insurgere paulatim, munia
senatus magistratuum legum in se trahere, nullo
adversante, cum ferocissimi per acies aut proscriptione
cecidissent, ceteri nobilium, quanto quis servitio
promptior, opibus et honoribus extollerentur ac novis
ex rebus aucti tuta et praesentia quam vetera et
```

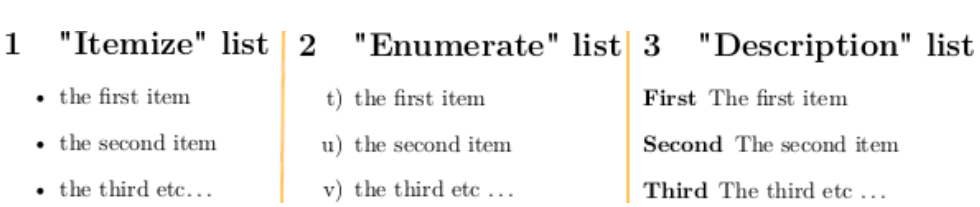
(continues on next page)

(continued from previous page)

```
periculosa mallent.
   Neque provinciae illum rerum statum abnuebant,
suspecto senatus populique imperio ob certamina
potentium et avaritiam magistratuum, invalido legum
auxilio quae vi ambitu postremo pecunia turbabantur.
'''

def main():
 cf.active = cf.Version1()
 doc = Document(data=NoEscape(lorem))
 doc.generate_pdf(
   'config1_with_indent', clean_tex=False)

 cf.active = cf.Version1(indent=False)
 doc = Document(data=NoEscape(lorem))
 doc.generate_pdf(
   'config2_without_indent', clean_tex=False)

 with cf.Version1().use():
  doc = Document(data=NoEscape(lorem))
  doc.generate_pdf(
    'config3_with_indent_again', clean_tex=False)

 doc = Document(data=NoEscape(lorem))
 doc.generate_pdf(
   'config4_without_indent_again', clean_tex=False)

if __name__ == '__main__':
    main()
```

8.14 Lists

This example shows list functionality.



The file lists.py:

```
# Test for list structures in PyLaTeX.
from pylatex import (
  Document, Section, Itemize, Enumerate, Description,
  Command, NoEscape)

doc = Document()

# create a bulleted "itemize" list like the below:
# \begin{itemize}
#   \item The first item
#   \item The second item
#   \item The third etc \ldots
# \end{itemize}

with doc.create(Section('"Itemize" list')):
  with doc.create(Itemize()) as itemize:
    itemize.add_item("the first item")
    itemize.add_item("the second item")
    itemize.add_item("the third etc")
    # you can append to existing items
    itemize.append(Command("ldots"))

# create a numbered "enumerate" list like the below:
# \begin{enumerate}[label=\alph*),start=20]
#   \item The first item
#   \item The second item
#   \item The third etc \ldots
# \end{enumerate}
```

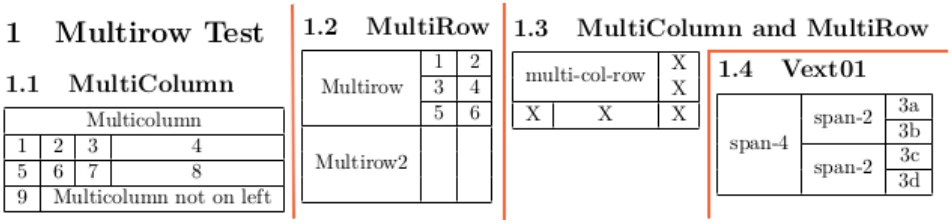
(continues on next page)

(continued from previous page)

```
with doc.create(Section('"Enumerate" list')):
 with doc.create(Enumerate(
   enumeration_symbol=r"\alph*)",
   options={'start': 20})) as enum:
  enum.add_item("the first item")
  enum.add_item("the second item")
  enum.add_item(NoEscape("the third etc \\ldots"))

# create a labelled "description" list like the below:
# \begin{description}
#   \item[First] The first item
#   \item[Second] The second item
#   \item[Third] The third etc \ldots
# \end{description}

with doc.create(Section('"Description" list')):
 with doc.create(Description()) as desc:
  desc.add_item("First", "The first item")
  desc.add_item("Second", "The second item")
  desc.add_item(
    "Third", NoEscape("The third etc \\ldots"))

doc.generate_pdf('lists', clean_tex=False)
```

8.15 Multirow

This example shows how multirow and multicolumns can be used.



The file multirow.py:

```
from pylatex import (
  Document, Section, Subsection, Tabular,
  MultiColumn, MultiRow)

doc = Document("multirow")
section = Section('Multirow Test')

test1 = Subsection('MultiColumn')
test2 = Subsection('MultiRow')
test3 = Subsection('MultiColumn and MultiRow')
test4 = Subsection('Vext01')

table1 = Tabular('|c|c|c|c|')
table1.add_hline()
table1.add_row((MultiColumn(4, align='|c|', data=
 ↪'Multicolumn'),))
table1.add_hline()
table1.add_row((1, 2, 3, 4))
table1.add_hline()
table1.add_row((5, 6, 7, 8))
table1.add_hline()
row_cells = ('9', MultiColumn(
 3, align='|c|',
 data='Multicolumn not on left'))
table1.add_row(row_cells)
table1.add_hline()
```

(continues on next page)

(continued from previous page)

```
table2 = Tabular('|c|c|c|')
table2.add_hline()
table2.add_row((MultiRow(3, data='Multirow'), 1, 2))
table2.add_hline(2, 3)
table2.add_row(('', 3, 4))
table2.add_hline(2, 3)
table2.add_row(('', 5, 6))
table2.add_hline()
table2.add_row((MultiRow(3, data='Multirow2'), '', ''))
table2.add_empty_row()
table2.add_empty_row()
table2.add_hline()

table3 = Tabular('|c|c|c|')
table3.add_hline()
table3.add_row((MultiColumn(
 2, align='|c|',
 data=MultiRow(2, data='multi-col-row')), 'X'))
table3.add_row((MultiColumn(
 2, align='|c|', data=''), 'X'))
table3.add_hline()
table3.add_row(('X', 'X', 'X'))
table3.add_hline()

table4 = Tabular('|c|c|c|')
table4.add_hline()
col1_cell = MultiRow(4, data='span-4')
col2_cell = MultiRow(2, data='span-2')
table4.add_row((col1_cell, col2_cell, '3a'))
table4.add_hline(start=3)
table4.add_row(('', '', '3b'))
table4.add_hline(start=2)
table4.add_row(('', col2_cell, '3c'))
```

(continues on next page)

(continued from previous page)

```
table4.add_hline(start=3)
table4.add_row(('', '', '3d'))
table4.add_hline()

test1.append(table1)
test2.append(table2)
test3.append(table3)
test4.append(table4)

section.append(test1)
section.append(test2)
section.append(test3)
section.append(test4)

doc.append(section)
doc.generate_pdf(clean_tex=False)
```

8.16 Full Report

This example demonstrates several features of PyLaTeX.

It includes plain equations, tables, equations using numpy objects, tikz plots, and figures.

1 The simple stuff

Some regular text and some*italic text*.
Also some crazy characters: $&#{}

1.1 Math that is incorrect

$$2 * 3 = 9$$

1.2 Table of something

1	2	3	4
4	5	6	7

2 The fancy stuff

2.1 Correct matrix equations

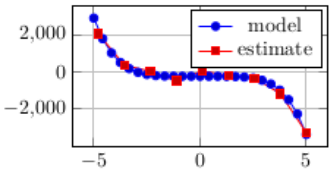
$$\begin{pmatrix} 2 & 3 & 4 \\ 0 & 0 & 1 \\ 0 & 0 & 2 \end{pmatrix} \begin{pmatrix} 100 \\ 10 \\ 20 \end{pmatrix} = \begin{pmatrix} 310 \\ 20 \\ 40 \end{pmatrix}$$

2.2 Alignat math environment

$$\frac{a}{b} = 0$$

$$\begin{pmatrix} 2 & 3 & 4 \\ 0 & 0 & 1 \\ 0 & 0 & 2 \end{pmatrix} \begin{pmatrix} 100 \\ 10 \\ 20 \end{pmatrix} = \begin{pmatrix} 310 \\ 20 \\ 40 \end{pmatrix}$$

2.3 Beautiful graphs



2.4 Cute kitten pictures

Figure 1: Look it's on its back

The file full.py:

```
import numpy as np

from pylatex import (
 Document, Section, Subsection, Tabular, Math, TikZ,
 Axis, Plot, Figure, Matrix, Alignat)
from pylatex.utils import italic
import os

image_filename = os.path.join(
  os.path.dirname(__file__), 'kitten.jpg')

geometry_options = {
```

(continues on next page)

(continued from previous page)

```
    "tmargin": "1cm",
    "lmargin": "10cm"}
doc = Document(geometry_options=geometry_options)

with doc.create(Section('The simple stuff')):
  doc.append('Some regular text and some')
  doc.append(italic('italic text. '))
  doc.append('\nAlso some crazy characters: $&#{}')
  with doc.create(
                  Subsection('Math that is incorrect')):
      doc.append(Math(data=['2*3', '=', 9]))

  with doc.create(Subsection('Table of something')):
   with doc.create(Tabular('rc|cl')) as table:
    table.add_hline()
    table.add_row((1, 2, 3, 4))
    table.add_hline(1, 2)
    table.add_empty_row()
    table.add_row((4, 5, 6, 7))

a = np.array([[100, 10, 20]]).T
M = np.matrix([[2, 3, 4],
               [0, 0, 1],
               [0, 0, 2]])

with doc.create(Section('The fancy stuff')):
 with doc.create(
                Subsection('Correct matrix equations')):
  doc.append(Math(data=[
    Matrix(M), Matrix(a), '=', Matrix(M * a)]))

 with doc.create(Subsection(
   'Alignat math environment')):
  with doc.create(Alignat(
```

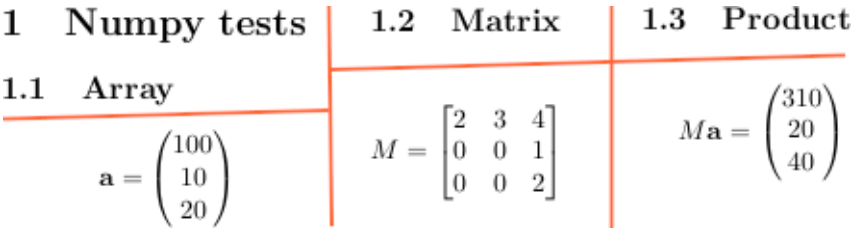
(continues on next page)

(continued from previous page)

```
    numbering=False, escape=False)) as agn:
   agn.append(r'\frac{a}{b} &= 0 \\')
   agn.extend([
    Matrix(M), Matrix(a), '&=', Matrix(M * a)])

 with doc.create(Subsection('Beautiful graphs')):
  with doc.create(TikZ()):
   plot_options = 'height=4cm, width=6cm, grid=major'
   with doc.create(Axis(options=plot_options)) as plot:
    plot.append(Plot(name='model', func='-x^5 - 242'))

    coordinates = [
        (-4.77778, 2027.60977),
        (-3.55556, 347.84069),
        (-2.33333, 22.58953),
        (-1.11111, -493.50066),
        (0.11111, 46.66082),
        (1.33333, -205.56286),
        (2.55556, -341.40638),
        (3.77778, -1169.24780),
        (5.00000, -3269.56775),
    ]

    plot.append(Plot(
     name='estimate', coordinates=coordinates))

 with doc.create(Subsection('Cute kitten pictures')):
  with doc.create(Figure(position='h!')) as kitten_pic:
   kitten_pic.add_image(image_filename, width='120px')
   kitten_pic.add_caption('Look it\'s on its back')

doc.generate_pdf('full', clean_tex=False)
```

8.17 Numpy

This example shows numpy functionality.



The file numpy_ex.py:

```
import numpy as np

from pylatex import (
  Document, Section, Subsection, Math, Matrix,
  VectorName)

a = np.array([[100, 10, 20]]).T

doc = Document()
section = Section('Numpy tests')
subsection = Subsection('Array')

vec = Matrix(a)
vec_name = VectorName('a')
math = Math(data=[vec_name, '=', vec])

subsection.append(math)
section.append(subsection)

subsection = Subsection('Matrix')
M = np.matrix([[2, 3, 4],
               [0, 0, 1],
```

(continues on next page)

(continued from previous page)

```
                [0, 0, 2]])
matrix = Matrix(M, mtype='b')
math = Math(data=['M=', matrix])

subsection.append(math)
section.append(subsection)

subsection = Subsection('Product')

math = Math(data=['M', vec_name, '=', Matrix(M * a)])
subsection.append(math)

section.append(subsection)

doc.append(section)
doc.generate_pdf('numpy_ex', clean_tex=False)
```

8.18 Environment

Wrapping existing LaTeX environments with the Environment class.

1 Wrapping Latex Environments

The following is a demonstration of a custom LaTeX command with a couple of parameters.

```
This is verbatim, alltt, text.
```

```
Setting escape to False ensures that text in the environment is not
subject to escaping...
```

```
Setting content_separator ensures that line endings are broken in
the latex just as they are in the input text.
alltt supports math: x^2 = 10
```

This is back to normal text...

The file environment_ex.py:

```
from pylatex.base_classes import Environment
from pylatex.package import Package
from pylatex import Document, Section
from pylatex.utils import NoEscape

class AllTT(Environment):
 """A class to wrap LaTeX's alltt environment."""

 packages = [Package('alltt')]
 escape = False
 content_separator = "\n"

# Create a new document
doc = Document()
with doc.create(Section(
        'Wrapping Latex Environments')):
 doc.append(NoEscape(
  r"""
  The following is a demonstration of a custom \LaTeX{}
  command with a couple of parameters.
  """))

 # Put some data inside the AllTT environment
 with doc.create(AllTT()):
  verbatim = (
   "This is verbatim, alltt, text.\n\n\n"
   "Setting \\underline{escape} to \\underline{False} "
   "ensures that text in the environment is not\n"
   "subject to escaping...\n\n\n"
   "Setting \\underline{content_separator} "
   "ensures that line endings are broken in\n"
   "the latex just as they are in the input text.\n"
   "alltt supports math: \\(x^2=10\\)")
```

(continues on next page)

(continued from previous page)

```
  doc.append(verbatim)

 doc.append("This is back to normal text...")

# Generate pdf
doc.generate_pdf('environment_ex', clean_tex=False)
```

8.19 os-walk-3.py

The file os-walk-3.py:

```
import os
for root, dirs, files in os.walk ("."):
   for name in files:
     extension = os.path.splitext (name) [1]
     if extension == ".py":
       print ("\n## " + name + "\n\n")
       print ("\nThe file " + name + "::\n\n")
       f = open (name)
       lines = f.readlines()
       for x in lines:
         print ("    " + x, end="")
       print ()
```

8.20 Quantities

1 Quantity tests

1.1 Scalars with units

$$F = 1.982 \times 10^{20}\,\mathrm{N}$$

1.2 Scalars without units

$$N = 7.4 \times 10^{9}$$

1.3 Scalars with uncertainties

$$A = (42.0 \pm 3.2)\,\mathrm{m}^2$$

This example shows quantities functionality.

The file quantities_ex.py:

```
import quantities as pq

from pylatex import (
  Document, Section, Subsection, Math, Quantity)

doc = Document()
section = Section('Quantity tests')

subsection = Subsection('Scalars with units')
G = pq.constants.Newtonian_constant_of_gravitation
moon_earth_distance = 384400 * pq.km
moon_mass = 7.34767309e22 * pq.kg
earth_mass = 5.972e24 * pq.kg
moon_earth_force =
    G * moon_mass * earth_mass / moon_earth_distance**2
q1 = Quantity(
  moon_earth_force.rescale(pq.newton),
  options={
    'round-precision': 4,
    'round-mode': 'figures'})
math = Math(data=['F=', q1])
```

(continues on next page)

(continued from previous page)

```
subsection.append(math)
section.append(subsection)

subsection = Subsection('Scalars without units')
world_population = 7400219037
N = Quantity(
  world_population,
  options={
    'round-precision': 2,
    'round-mode': 'figures'},
  format_cb="{0:23.17e}".format)
subsection.append(Math(data=['N=', N]))
section.append(subsection)

subsection = Subsection('Scalars with uncertainties')
width = pq.UncertainQuantity(7.0, pq.meter, .4)
length = pq.UncertainQuantity(6.0, pq.meter, .3)
area = Quantity(
  width * length,
  options='separate-uncertainty',
  format_cb=lambda x: "{0:.1f}".format(float(x)))
subsection.append(Math(data=['A=', area]))
section.append(subsection)

doc.append(section)
doc.generate_pdf('quantities_ex', clean_tex=False)
```

8.21 Own Commands

How to represent your own LaTeX commands and environments in PyLaTeX.

The file own_commands_ex.py:

```
from pylatex.base_classes import (
  Environment, CommandBase, Arguments)
from pylatex.package import Package
from pylatex import Document, Section, UnsafeCommand
from pylatex.utils import NoEscape


class ExampleEnvironment(Environment):
    """
    A class representing a custom LaTeX environment.

    This class represents a custom LaTeX environment
    named ``exampleEnvironment``.
    """

    _latex_name = 'exampleEnvironment'
    packages = [Package('mdframed')]


class ExampleCommand(CommandBase):
    """
```

(continues on next page)

(continued from previous page)

```
    A class representing a custom LaTeX command.

    This class represents a custom LaTeX command named
    ``exampleCommand``.
    """

    _latex_name = 'exampleCommand'
    packages = [Package('color')]


# Create a new document
doc = Document()
with doc.create(Section('Custom commands')):
 doc.append(NoEscape(
  r"""
  The following is a demonstration of a custom \LaTeX{}
  command with a couple of parameters.
  """))

 # Define the new command
 new_comm = UnsafeCommand(
   'newcommand', '\exampleCommand', options=3,
   extra_arguments=r'\color{#1} #2 #3 \color{black}')
 doc.append(new_comm)

 # Use our new command with different arguments
 doc.append(ExampleCommand(
  arguments=Arguments('blue', 'Hello', 'World!')))
 doc.append(ExampleCommand(
  arguments=Arguments('green', 'Hello', 'World!')))
 doc.append(ExampleCommand(
  arguments=Arguments('red', 'Hello', 'World!')))

with doc.create(Section('Custom environments')):
```

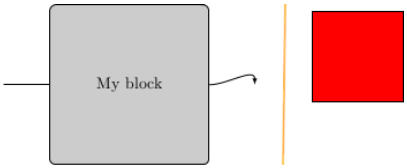
(continues on next page)

(continued from previous page)

```
 doc.append(NoEscape(
  r"""
  The following is a demonstration of a custom \LaTeX{}
  environment using the mdframed package.
  """))

 # Define a style for our box
 mdf_style_definition = UnsafeCommand('mdfdefinestyle',
   arguments=['my_style',
     ('linecolor=#1,'
      'linewidth=#2,'
      'leftmargin=1cm,'
      'leftmargin=1cm')])

 # Define a new environment by style definition above
 new_env = UnsafeCommand(
     'newenvironment', 'exampleEnvironment', options=2,
     extra_arguments=[
         mdf_style_definition.dumps() +
         r'\begin{mdframed}[style=my_style]',
         r'\end{mdframed}'])
 doc.append(new_env)

 # Usage of the newly created environment
 with doc.create(ExampleEnvironment(
   arguments=Arguments('red', 3))) as environment:
  environment.append('This is the actual content')

# Generate pdf
doc.generate_pdf('own_commands_ex', clean_tex=False)
```

8.22 Tikzdraw

This example shows TikZ drawing capabilities.



The file tikzdraw.py:

```
from pylatex import (
  Document, TikZ, TikZNode, TikZDraw, TikZCoordinate,
  TikZUserPath, TikZOptions)

# create document
doc = Document()

# add our sample drawings
with doc.create(TikZ()) as pic:

 # options for our node
 node_kwargs = {
  'align': 'center', 'minimum size': '100pt',
  'fill': 'black!20'}

 # create our test node
 box = TikZNode (
   text='My block', handle='box',
   at=TikZCoordinate(0, 0),
   options=TikZOptions (
     'draw', 'rounded corners', **node_kwargs))

 # add to tikzpicture
 pic.append(box)
```

(continues on next page)

(continued from previous page)

```
 # draw a few paths
 pic.append(TikZDraw(
   [TikZCoordinate(0, -6),
    'rectangle',
    TikZCoordinate(2, -8)],
   options=TikZOptions(fill='red')))

 # show use of anchor, relative coordinate
 pic.append(TikZDraw([
   box.west, '--', '++(-1,0)']))

 # demonstrate the use of the with syntax
 with pic.create(TikZDraw()) as path:

     # start at an anchor of the node
     path.append(box.east)

     # necessary here because 'in' is a python keyword
     path_options = {'in': 90, 'out': 0}
     path.append(
       TikZUserPath('edge',
         TikZOptions('-latex', **path_options)))
     path.append(TikZCoordinate(1, 0, relative=True))

doc.generate_pdf('tikzdraw', clean_tex=False)
```

8.23 Minipage

This example shows the functionality of the MiniPage element.

It creates a sample page filled with labels using the MiniPage element.

The file minipage.py:

```
from pylatex import (
  Document, MiniPage, LineBreak, VerticalSpace, config)

def generate_labels():
 config.active = config.Version1 (indent=False)
 geometry_options = {"margin": "0.5in"}
 doc = Document(geometry_options=geometry_options)

 doc.change_document_style("empty")
 for i in range(10):
   with doc.create(MiniPage(width=r"0.5\textwidth")):
     doc.append("Vladimir Gorovikov")
     doc.append("\n")
     doc.append("Company Name")
     doc.append("\n")
     doc.append("Somewhere, City")
     doc.append("\n")
     doc.append("Country")
   if (i % 2) == 1:
     doc.append(VerticalSpace("20pt"))
     doc.append(LineBreak())

 doc.generate_pdf("minipage", clean_tex=False)
generate_labels()
```

PYTHON MODULE INDEX

m

INDEX

L

M

N

P

R

S

T