

課長のせいで今日も死んでます。 楽して覚える情報技術

[著] 株式会社おもしろテクノロジー

技術書典 18（2025 年夏）新刊
2025 年 5 月 30 日 ver 1.0

■免責

本書は情報の提供のみを目的としています。

本書の内容を実行・適用・運用したことで何が起こりようとも、それは実行・適用・運用した人自身の責任であり、著者や関係者はいかなる責任も負いません。

■商標

本書に登場するシステム名や製品名は、関係各社の商標または登録商標です。

また本書では、™、®、©などのマークは省略しています。

まえがき / はじめに

***** 注意 *****

このドキュメントは、Re:VIEW のサンプル原稿を兼ねています。自分の原稿を書くときは、サンプルの原稿ファイルと画像ファイルを消してください。

▼ サンプルファイル（原稿と画像）の消し方（コマンドラインを知らない人はごめん！）

```
$ ls contents/           ←原稿ファイルの一覧
00-preface.re           03-syntax.re           06-bestpractice.re
01-install.re           04-customize.re        92-filelist.re
02-tutorial.re          05-faq.re              99-postface.re
$ rm contents/*.re       ←原稿のファイルを消す
$ rm -r images/*         ←画像ファイル（が入ったディレクトリ）も消す
$ vi catalog.yml         ←各章のファイル名を変更する
```

本文での、1 行あたりの文字数の確認：
一二三四五六七八九十一二三四五六七八九十一二三四五六七八九十一二三四
五六七八九十一二三四五六七八九十

▼ プログラムリストでの、1 行あたりの文字数の確認（font: beramono）

```
123456789_123456789_123456789_123456789_123456789_123456789_123456789_123>
>456789_123456789_123456789_
一二三四五六七八九十一二三四五六七八九十一二三四五六七八九十一二>
>三四五六七八九十
```

▼ ターミナルでの、1 行あたりの文字数の確認（font: inconsolata）

```
123456789_123456789_123456789_123456789_123456789_123456789_123456789_123456789>
>_123456789_123456789_
一二三四五六七八九十一二三四五六七八九十一二三四五六七八九十一二三>
>四五五六七八九十
```

***** 以下、まえがきのサンプル *****

本書を手にとっていただき、ありがとうございます。
本書は、XXX についてわかりやすく解説した本です。この本を読めば、XXX の基本が身につきます。

本書の目的

本書の目的は、XXX の基礎的な使い方を身につけることです。具体的には、XXX を使って XXX や XXX や XXX ができるようになることです。

ただし、XXX や XXX といった内容は、本書では扱いません。

本書の対象読者

本書では次のような人を対象としています。

- XXX について興味がある人
- XXX の入門書を読んだ初級者の人（まったくの初心者は対象外です）

前提とする知識

本書を読むにあたり、次のような知識が必要となります。

- Linux についての基礎知識
- 何らかのプログラミング言語の基礎知識

問い合わせ先

本書に関する質問やお問い合わせは、次のページまでお願いします。正誤表とサンプルコードもここにあります。

- URL: <https://www.example.com/mybook/>

謝辞

本書は XXXX 氏と XXXX 氏にレビューしていただきました。この場を借りて感謝します。ありがとうございました。

凡例

本書では、プログラムコードを次のように表示します。太字は強調を表します。

```
print("Hello, world!\\n");
```

 ←太字は強調

プログラムコードの差分を表示する場合は、追加されたコードを太字で、削除されたコードを取り消し線で表します。

```
print("Hello, world!\\n";);
```

 ←取り消し線は削除したコード

```
print("Hello, \"+name+\"!\\n");
```

 ←太字は追加したコード

長い行が右端で折り返されると、折り返されたことを表す小さな記号がつきます。

```
123456789_123456789_123456789_123456789_123456789_123456789_123456789_123>  
>456789_123456789_
```

ターミナル画面は、次のように表示します。行頭の「\$」はプロンプトを表し、ユーザが入力するコマンドには薄い下線を引いています。

```
$ echo Hello
```

 ←行頭の「\$」はプロンプト、それ以降がユーザ入力

本文に対する補足情報や注意・警告は、次のようなノートや囲み枠で表示します。

.....
ノートタイトル
ノートは本文に対する補足情報です。
.....



タイトル

本文に対する補足情報です。



タイトル

本文に対する注意・警告です。

目次

まえがき / はじめに	i
第 1 章 インストール	1
1.1 Ruby と TeXLive のインストール	1
Docker の使い方を知っている場合	1
macOS を使っている場合	2
Windows を使っている場合	4
1.2 プロジェクトを作成	6
1.3 サンプルの PDF ファイルを生成	7
Docker の場合	7
macOS の場合	8
Windows の場合	8
1.4 注意点	10
第 2 章 チュートリアル	11
2.1 用語の説明	11
2.2 フォルダとファイルの説明	13
2.3 原稿の追加と変更	15
既存の原稿ファイルを変更する	15
新しい原稿ファイルを追加する	16
「catalog.yml」の説明	18
コンパイルエラーになったら	19
文章をチェックする	19
2.4 基本的な記法	20
コメント	20
段落と改行	20
見出し	21
箇条書き	21
用語リスト	23
太字と強調	23
等幅フォント	24
プログラムリスト	24
ターミナル画面	26
脚注	26

図	26
ノート	27
表	29
数式	29
会話形式	30
用語、索引	31
単語展開	32
2.5 印刷用 PDF と電子用 PDF	33
印刷用と電子用を切り替えて PDF を生成する	33
印刷用 PDF と電子用 PDF の違い	33
2.6 高速プレビュー	35
指定した章だけをコンパイルする	35
コンパイル回数を 1 回だけに制限する	35
画像読み込みを省略するドラフトモード	36
自動リロードつき HTML プレビュー	36
第 3 章 記法	39
3.1 コメント	39
行コメント	39
範囲コメント	39
3.2 段落と改行と空行	41
段落	41
改行	42
強制改行	43
強制空行	43
改段落	44
改ページ	44
3.3 見出し	46
見出しレベル	46
見出し用オプション	47
章の概要と著者名	47
章 ID	48
章を参照する	48
節や項を参照する	49
まえがき、あとがき、付録	50
部	51
段見出しと小段見出し	51
3.4 箇条書きと定義リスト	54
番号なし箇条書き	54
番号つき箇条書き	54
定義リスト	55

	説明リスト	57
3.5	インライン命令とブロック命令	61
	インライン命令	61
	ブロック命令	62
3.6	強調と装飾	63
	強調	63
	太字	63
	装飾	64
	文字サイズ	64
	マーキング用	64
	その他のインライン命令	65
3.7	プログラムリスト	66
	基本的な書き方	66
	リスト番号	66
	ラベルの自動指定	67
	長い行の折り返し	67
	改行文字を表示	69
	インデント	69
	外部ファイルの読み込み	70
	タブ文字	71
	全角文字の幅を半角 2 文字分にする	71
	出力結果	73
	その他のオプション	74
3.8	行番号	75
	プログラムリストに行番号をつける	75
	行番号の幅を指定する	76
	複雑な行番号指定	77
3.9	ターミナル	79
	基本的な書き方	79
	ユーザ入力	80
	ターミナルのカーソル	80
3.10	脚注	81
	脚注の書き方	81
	脚注の注意点	81
3.11	ノート	83
	ノートの書き方	83
	ラベルをつけて参照する	84
	ノート以外のミニブロック	85
3.12	コラム	88
	コラムとは	88
	コラムの書き方	88

	[コラム] サンプルコラム 1	88
	コラム内の脚注	89
	[コラム] サンプルコラム 2	89
	コラムを参照	89
	[コラム] サンプルコラム 3	90
	コラムの制限	90
3.13	画像	91
	画像ファイルの読み込み方	91
	画像の表示幅を指定する	91
	章ごとの画像フォルダ	93
	画像のまわりに線をつける	93
	画像の配置位置を指定する	93
	画像に番号も説明もつけない	94
	文章の途中に画像を読み込む	95
	画像とテキストを並べて表示する	96
3.14	表	98
	表の書き方	98
	ラベルの自動採番	98
	右寄せ、左寄せ、中央揃え	99
	縦の罫線	100
	横の罫線	101
	セル幅	102
	空欄	103
	表示位置の指定	104
	表のフォントサイズ	105
	CSV 形式の表	106
	外部ファイルを読み込む	107
	ヘッダの行数を指定する	108
	ヘッダのカラム数を指定する	109
	表を画像で用意する	110
3.15	数式	112
	数式の書き方	112
	数式を文中に埋め込む	112
	数式のフォント	112
3.16	会話形式	114
	アイコン画像を使う場合	114
	アイコン画像を使わない場合	115
	コンパクトな書き方	116
	短縮名	117
	カスタマイズ	118
3.17	用語、索引	121

	索引機能をオンにする	121
	用語の書き方	122
	親子関係にある用語	123
	親子関係の順番を入れ替える	123
	転送先の用語を指定する	124
	よみがなの再利用	125
	よみがな辞書を使う	125
	用語と索引の補足事項	126
	Re:VIEW との違い	126
3.18	単語展開	128
	単語展開用のインライン命令	128
	単語展開用の辞書ファイル	128
	Re:VIEW との違い	129
3.19	その他	130
	URL、リンク	130
	引用	131
	傍点	131
	ルビ、読みがな	131
	何もしないコマンド	132
	コード中コメント	132
	特殊文字	133
	右寄せ、センタリング	133
	縦方向の空き（実験的機能）	134
	生データ	134
	参考文献	135
第 4 章	カスタマイズ	137
4.1	命令	137
	新しいインライン命令を追加する	137
	新しいブロック命令を追加する	138
4.2	ページと本文	141
	[PDF] ページサイズを B5 や A5 に変更する	141
	[PDF] 本文の幅を指定する	141
	[PDF] 本文の高さを指定する	141
	[PDF] 塗り足しをつける	142
	[PDF] 塗り足しを可視化する	143
	[PDF] 章の右ページ始まりをやめる	143
	[PDF] 「//info」や「//warning」のデザインを変更する	143
4.3	フォント	145
	[PDF] フォントサイズを変更する	145
	[PDF] フォントの種類を変更する	145

4.4	プログラムコード	147
	[PDF] 説明文直後での改ページを防ぐ	147
	[PDF] プログラムコードのフォントを変更する	148
	プログラムコードの枠線	148
	オプションのデフォルト値	148
	[PDF] 折り返し記号や行番号をマウス選択の対象から外す	149
4.5	見出し	151
	章のタイトルデザインを変更する	151
	節のタイトルデザインを変更する	151
	段見出しのデザインを変更する	151
	[PDF] 章や節のタイトルを独自にデザインする	153
	[PDF] 章扉をつける	153
	[PDF] 章扉に背景色をつける	154
	[PDF] 節ごとに改ページする	154
	項に番号をつける	155
	項を目次を含める	155
	[PDF] 項への参照に節タイトルを含める	155
	項タイトルの記号を外す	156
	[PDF] 項タイトルの記号をクローバーから変更する	156
4.6	本	157
	本のタイトルや著者名を変更する	157
	[PDF] 表紙や大扉や奥付となる PDF ファイルを指定する	157
	[PDF] 表紙の PDF から塗り足しを除いて取り込む	158
	[PDF] 大扉のデザインを変更する	159
	[PDF] 奥付のデザインを変更する	159
	[PDF] 奥付を改ページしない	159
	[PDF] 奥付に発行者と連絡先を記載する	160
	[PDF] 注意事項の文章を変更する	161
	[PDF] 注意事項の文章を出さない	161
4.7	索引	162
	[PDF] 用語のグループ化を文字単位にする	162
	[PDF] 連続したピリオドのかわりに空白を使う	162
	[PDF] 用語見出しのデザインを変更する	162
	[PDF] その他	163
4.8	Rake タスク	164
	デフォルトタスクを設定する	164
	独自のタスクを追加する	164
	コンパイルの前処理を追加する	164
第 5 章	FAQ	165
5.1	コンパイルエラー	165

	コンパイルエラーが出たとき	165
	「review-pdfmaker」で PDF が生成できない	166
5.2	本文	167
	左右の余白が違う	167
	文章中のプログラムコードに背景色をつけたい	167
	索引をつけたい	167
5.3	プログラムコードとターミナル	168
	プログラムコードの見た目が崩れる	168
	右端に到達してないのに折り返しされる	168
	全角文字の幅を半角文字 2 個分にしたい	169
	ターミナルの出力を折り返しせずに表示したい	169
	コードハイライトしたい	170
5.4	画像	171
	PDF と HTML とで画像の表示サイズを変えたい	171
	印刷用 PDF と電子用 PDF とで異なる解像度の画像を使いたい	171
	画像の解像度はどのくらいにすればいいか	172
	白黒印刷用の PDF にカラー画像を使っても大丈夫か	173
5.5	ノンブル	174
	ノンブルとは	174
	ノンブルを必要とする印刷所	174
	ノンブルのつけ方	174
	ノンブルの調整	176
	ノンブルの注意点	176
5.6	L ^A T _E X 関連	178
	LuaLaTeX と jlreq を使いたい	178
	L ^A T _E X のスタイルファイルから環境変数を参照する	178
5.7	その他	179
	高度なカスタマイズ	179
第 6 章	よりよい本づくり	181
6.1	まえがき	181
	まえがきには章番号をつけない	181
	まえがきに「本書の目的」を入れる	181
	まえがきに「対象読者」を入れる	182
	まえがきに「前提知識」を入れる	182
	まえがきにサポートサイトの URL を載せる	183
	まえがきに謝辞を載せる	183
	まえがきにソフトウェアのバージョンを載せる	183
	まえがきの章タイトル以外は目次に載せない	184
6.2	初心者向け入門書	185
	初心者向け入門書ではフォントを大きめにする	185

	初心者向け入門書では節と項の見分けをつきやすくする	185
	初心者向け入門書では節ごとに改ページする	185
6.3	箇条書き	187
	箇条書きを正しく使い分ける	187
	箇条書き直後に継続する段落は字下げしない	188
6.4	本文	190
	強調箇所は太字のゴシック体にする	190
	強調と傍点を使い分ける	190
	[PDF] 記号と日本語はくっつくことに注意する	191
	[PDF] 等幅フォントで余計な空白が入るのを防ぐ	191
	文章中のコードは折り返しする	193
	ノートとコラムを使い分ける	193
6.5	見出し	194
	節タイトルが複数行になるなら下線や背景色を使わない	194
	項を参照するなら項番号をつける	194
6.6	プログラムコード	196
	0 と O や 1 と l が見分けやすいフォントを使う	196
	フォントを小さくしすぎない	196
	重要箇所を目立たせる	197
	重要でない箇所を目立たせない	198
	差分（追加と削除）の箇所を明示する	198
	長い行の折り返し箇所を明示する	199
	行番号を控えめに表示する	200
	行番号を考慮して長い行を折り返す	200
	ページまたぎしていることを可視化する	201
	インデントを可視化する	201
6.7	図表	203
	図表が次のページに送られてもスペースを空けない	203
	大きい図表は独立したページに表示する	203
	図表は番号で参照する	203
	白黒印刷でも問題ないか確認する	203
	表のカラム幅を指定する	203
	表のカラムが数値なら右寄せにする	204
6.8	ページデザイン	206
	[PDF] 印刷用では左右の余白幅を充分にとる	206
	[PDF] タブレット向けでは余白を切り詰める	207
	[PDF] 電子用 PDF ではページ番号の位置を揃える	207
	非 Retina 向けには本文をゴシック体にする	207
6.9	大扉	209
	長いタイトルでは改行箇所を明示する	209
	大扉を別のソフトで作成する	209

6.10	奥付	211
	奥付は最後のページに置く	211
	奥付に更新履歴とイベント名を記載する	211
	利用した素材の作者と URL を奥付に記載する	211
	利用したソフトウェアを奥付に記載する	211
6.11	textlint による文章チェック	212
	textlint とは	212
	textlint の注意点	213
	Starter のプロジェクトで textlint を使う	215
	Starter における textlint 実行の仕組み	216
付録 A	Re:VIEW との差分	217
A.1	まとめ記事	217
A.2	プロジェクト作成	218
A.3	コメント	219
A.4	箇条書き	220
A.5	定義リスト	221
A.6	見出し（章、節、項、目）	222
A.7	インライン命令	223
A.8	ブロック命令	224
A.9	プログラムコード	225
A.10	ターミナル画面	227
A.11	ノート、ミニブロック	228
A.12	図	229
A.13	表	230
A.14	会話形式	231
A.15	用語、索引	232
A.16	表紙、大扉、奥付	233
A.17	PDF ファイル	234
A.18	L ^A T _E X 関連	235
A.19	エラメッセージ	237
A.20	その他	238
付録 B	ファイルとフォルダ	239
付録 C	開発の背景	243
	あとがき / おわりに	251

第 1 章

インストール

Re:VIEW Starter を使っていただき、ありがとうございます。

この章では、Re:VIEW Starter を使うために必要となる「Ruby」と「TeXLive」のインストール方法を説明します。

1.1 Ruby と TeXLive のインストール

Re:VIEW Starter で PDF を生成するには、Ruby と TeXLive のインストールが必要です。

- Ruby とは世界中で人気のあるプログラミング言語のことです。原稿ファイル（「*.re」ファイル）を読み込むのに必要です。
- TeXLive とは有名な組版ソフトのひとつです。PDF ファイルを生成するのに必要です。

これらのインストール手順を説明します。

Docker の使い方を知っている場合

Docker とは、簡単に言えば「Windows や Mac で Linux コマンドを実行するためのソフトウェア」です*1。

Docker の使い方が分かる人は、`kauplan/review2.5`*2 の Docker イメージを使ってください。これで Ruby と TeXLive の両方が使えるようになります。

▼ Docker イメージのダウンロードと動作確認

```
$ docker pull kauplan/review2.5    ← 3GB 以上ダウンロードするので注意
$ docker run --rm kauplan/review2.5 ruby --version
ruby 2.5.1p57 (2018-03-29 revision 63029) [x86_64-linux-gnu]
$ docker run --rm kauplan/review2.5 uplatex --version
e-upTeX 3.14159265-p3.7.1-u1.22-161114-2.6 (utf8.uptex) (TeX Live 2017/Debian)
```

*1 Docker についてのこの説明は、技術的にはもっとも正確ではありません。しかし IT エンジニア以外に説明するには、このような説明で充分です。

*2 <https://hub.docker.com/r/kauplan/review2.5/>

...(省略)...

なお macOS のように Ruby が使える環境なら、「`rake setup:dockerimage`」を実行すると自動的に「`docker pull kauplan/review2.5`」が実行されます。

macOS を使っている場合

macOS において、Ruby と TeXLive をインストールする方法を説明します。

Ruby のインストール

macOS には最初から Ruby がインストールされているため、Ruby を別途インストールする必要はありません。

Ruby がインストールされていることを確認するために、Terminal.app^{*3}を起動して以下のコマンドを入力してみましょう。なお各行の先頭にある「\$」は入力せず、下線がついている部分のコマンドを入力してください。

▼ Ruby がインストールされていることを確認

```
$ which ruby      ←下線が引かれたコマンドだけを入力すること
/usr/bin/ruby     ←このような表示になるはず
$ ruby --version  ←下線が引かれたコマンドだけを入力すること
ruby 2.3.7p456 (2018-03-28 revision 63024) [universal.x86_64-darwin18]
```

実際の出力結果は上と少し違うはずですが、だいたい合っていれば OK です。

また必要なライブラリをインストールするために、以下のコマンドも実行してください（各行の先頭にある「\$」は入力せず、それ以降のコマンドを入力してください）。

▼ 必要なライブラリをインストール

```
### Rubyに詳しくない初心者向け
$ rake setup:rubygems

### Rubyに詳しい人向け
$ gem install review --version=2.5
$ review version
2.5.0   ←必ず「2.5.0」であること（より新しいバージョンは未サポート）
```

..... macOS 標準の Ruby では「gem install」がエラーになる

macOS には標準で Ruby がインストールされていますが、それを使って「gem install」を実行するとエラーになります。

^{*3} Terminal.app は、ファインダで「アプリケーション」フォルダ > 「ユーティリティ」フォルダ > 「ターミナル」をダブルクリックすると起動できます。

▼ macOS 標準の Ruby では gem がインストールできない

```
$ gem install review --version=2.5
ERROR: While executing gem ... (Gem::FilePermissionError)
       You don't have write permissions for the /Library/Ruby/Gems/2.3.0 directory.
```

このエラーを回避するには、次のような方法があります。

- (a) Homebrew などを使って別の Ruby をインストールする。
- (b) ~/.gemrc に「install: --user-install」という行を追加する。
- (c) 「rubygems/」のようなフォルダを作り、そのパスを環境変数\$GEM_HOME に設定する。

しかし Ruby やコマンドラインに慣れていない初心者にとっては、どの方法も難しいでしょう。

そこで、(c) に相当する作業を自動的に行えるようにしました。それが、Ruby に詳しくない初心者向けのコマンド「rake setup:rubygems」です。詳しいことは lib/tasks/starter.rake 中の「task :rubygems」の定義を見てください。また環境変数\$GEM_HOME を設定するかわりに、Rakefile の冒頭で Gem.path を設定しています。

.....

MacTeX のインストール

次に、MacTeX をダウンロードします。MacTeX とは、macOS 用の TeXLive です。MacTeX はサイズが大きい（約 4GB）ので、本家ではなく以下のミラーサイトのどれかを使ってください。

- <http://ftp.jaist.ac.jp/pub/CTAN/systems/mac/mactex/>
 > mactex-20200407.pkg
- <http://ftp.kddilabs.jp/pub/ctan/systems/mac/mactex/>
 > mactex-20200407.pkg
- <http://ftp.riken.go.jp/pub/CTAN/systems/mac/mactex/>
 > mactex-20200407.pkg

ダウンロードができれば、ダブルクリックしてインストールしてください。

インストールしたら Terminal.app で次のコマンドを入力し、動作を確認してください（各行の先頭にある「\$」は入力せず、それ以降のコマンドを入力してください）。

▼ MacTeX のインストールができたことを確認

```
$ which uplatex      ←下線が引かれたコマンドだけを入力すること
/Library/TeX/texbin/uplatex
$ uplatex --version  ←下線が引かれたコマンドだけを入力すること
e-upTeX 3.14159265-p3.8.1-u1.23-180226-2.6 (utf8.uptex) (TeX Live 2020)
... (省略) ...
```

実際の出力結果は上と少し違うはずですが、だいたい合っていれば OK です。

最後に、MacTeX でヒラギノフォントを使うための準備が必要です（これをしないと、PDF ファイルの日本語フォントが見るに耐えません）。そのためには、以下のページから「Bibunsho7-

patch-1.5-20200511.dmg」*4をダウンロードしてください。

- <https://github.com/munepi/bibunsho7-patch/releases>

ダウンロードしたらダブルクリックして解凍し、「Patch.app」をダブルクリックしてください。

Windows を使っている場合

Windows において、Ruby と TeXLive をインストールする方法を説明します。

Ruby のインストール

以下のページにアクセスし、「Ruby+Devkit 2.7.3-1」をダウンロードしてインストールしてください。x64 版と x86 版の 2 つがあるので、お使いの Windows が 64bit 版なら x64 版を、32bit 版または不明なら x86 版をダウンロードしてください。

- <https://rubyinstaller.org/downloads/>

インストールしたら、コマンドプロンプトで以下のコマンドを入力し、Ruby が実行できることを確かめてください。

▼ Ruby が実行できることを確認

```
C:\Users\yourname> ruby --version ←「ruby --version」だけを入力
ruby 2.7.3-1 [x64-mingw32]
```

実際の出力結果は上と少し違うでしょうが、だいたい合っていれば OK です。

また以下のコマンドを実行し、必要なライブラリをインストールします。

▼ 必要なライブラリをインストール

```
C:\Users\yourname> gem install review --version=2.5
C:\Users\yourname> review version
2.5.0 ←必ず「2.5.0」であること（より新しいバージョンは未サポート）
```

TeXLive のインストール

次に、Windows 用の TeXLive をインストールします。詳しくはこちらのページを参照してください。このうち、手順 (16) において右上の「すべて」ボタンを押してください（つまりすべてのパッケージを選ぶ）。

- 「TeXLive2020 をインストールして LaTeX を始める」

<https://tm23forest.com/contents/texlive2020-install-latex-begin>

インストールできたら、コマンドプロンプトで以下のコマンドを実行してみてください。

*4 日付が最新のものを選んでください。2020 年 6 月時点では「20200511」が最新です。

▼ コマンドが実行できることを確認

```
C:\Users\yourname> uplatex --version ←「uplatex --version」だけを入力
e-upTeX 3.14159265-p3.7.1-u1.22-161114-2.6 (utf8.uptex) (TeX Live 2020/W32TeX)
... (以下省略) ...
```

実際の出力結果は上と少し違うでしょうが、だいたい合っていれば OK です。

1.2 プロジェクトを作成

Ruby と TeXLive をインストールしたら、次に本を作るための「プロジェクト」を作成しましょう。

以下の Web サイトにアクセスしてください。

- <https://kauplan.org/reviewstarter/>

アクセスしたら、画面の指示に従って操作をしてください（詳細は省略します）。するとプロジェクトが作成されて、zip ファイルがダウンロードできます。

以降では、プロジェクトの zip ファイル名が「mybook.zip」だったとして説明します。

1.3 サンプルの PDF ファイルを生成

プロジェクトの zip ファイルをダウンロードしたら、解凍してサンプルの PDF ファイルを生成してみましょう。

Docker の場合

Docker を使う場合は、次のような手順でサンプルの PDF ファイルを生成してみましょう。

▼ Docker を使って PDF ファイルを生成する

```
$ unzip mybook.zip          ← zipファイルを解凍
$ cd mybook/                ←ディレクトリを移動
$ docker run --rm -v $PWD:/work -w /work kauplan/review2.5 rake pdf
$ ls *.pdf                  ← PDFファイルが生成できたことを確認
mybook.pdf
```

これで PDF ファイルが生成されるはずです。生成できなかった場合は、Twitter で「#review-starter」タグをつけて質問してください（相手先不要）。

Docker を使って PDF ファイルが作成できることを確認したら、このあとは docker コマンドを使わずとも「rake docker:pdf」だけで PDF ファイルが生成できます。

▼ より簡単に PDF ファイルを生成する

```
$ docker run --rm -v $PWD:/work -w /work kauplan/review2.5 rake pdf
$ rake docker:pdf          ←これだけでPDFファイルが生成される
```

もっと簡単に PDF ファイルを生成するための Tips

環境変数「\$RAKE_DEFAULT」を設定すると、引数なしの「rake」コマンドだけで PDF ファイルが生成できます。

▼ 環境変数「\$RAKE_DEFAULT」を設定する

```
$ export RAKE_DEFAULT="docker:pdf" ←環境変数を設定
$ rake docker:pdf
$ rake                             ←「rake docker:pdf」が実行される
```

またシェルのエイリアス機能を使うと、コマンドを短縮名で呼び出せます。

▼ シェルのエイリアス機能を使う

```
$ alias pdf="rake docker:pdf" ←エイリアスを設定
$ pdf                        ←「rake docker:pdf」が実行される
```

macOS の場合

macOS を使っている場合は、Terminal.app^{*5}を開いて以下のコマンドを実行してください。ここで、各行の先頭にある「\$」は入力せず、それ以降のコマンドを入力してください。

▼ PDF ファイルを生成する

```
$ unzip mybook.zip      ← zipファイルを解凍し、
$ cd mybook/           ←ディレクトリを移動
$ rake pdf              ← PDFファイルを生成
$ ls *.pdf              ← PDFファイルが生成されたことを確認
mybook.pdf
$ open mybook.pdf       ← PDFファイルを開く
```

これで PDF ファイルが生成されるはずです。生成できなかった場合は、Twitter で「#review-starter」タグをつけて質問してください（相手先不要）。

もっと簡単に PDF ファイルを生成するための Tips

環境変数「\$RAKE_DEFAULT」を設定すると、引数なしの「rake」コマンドだけで PDF ファイルが生成できます。

▼環境変数「\$RAKE_DEFAULT」を設定する

```
$ export RAKE_DEFAULT="pdf" ←環境変数を設定
$ rake pdf
$ rake                       ←「rake pdf」が実行される
```

またシェルのエイリアス機能を使うと、コマンドを短縮名で呼び出せます。

▼シェルのエイリアス機能を使う

```
$ alias pdf="rake pdf"      ←エイリアスを設定
$ pdf                       ←「rake pdf」が実行される
```

Windows の場合

Windows を使っている場合は、まずプロジェクトの zip ファイル（mybook.zip）をダブルクリックして解凍してください。

そしてコマンドプロンプトで以下のコマンドを実行してください。

▼ PDF ファイルを生成する

```
C:\Users\yourname> cd mybook      ←解凍してできたフォルダに移動
C:\Users\yourname\mybook> rake pdf ← PDFファイルを生成
C:\Users\yourname\mybook> dir *.pdf ← PDFファイルが生成されたことを確認
```

^{*5} 繰り返しになりますが、Terminal.app はファインダーで「アプリケーション」フォルダ > 「ユーティリティ」フォルダ > 「ターミナル」をダブルクリックすると起動できます。

これで PDF ファイルが生成されるはずです。生成できなかった場合は、Twitter で「**#review-starter**」タグをつけて質問してください（相手先不要）。

1.4 注意点

エラーが発生したときは、「コンパイルエラーになったら」(p.19)の説明を参考にしてください。
そのほか、次の点に注意してください。

- HTML ファイルを生成するときは「rake web」(または「rake docker:web」)を実行してください。
- ePub ファイルを生成するときは「rake epub」(または「rake docker:epub」)を実行してください。
- Markdown ファイルを生成するときは「rake markdown」(または「rake docker:markdown」)を実行してください。
- Starter では「review-pdfmaker」コマンドや「review-epubmaker」コマンドが使いません(実行はできますが望ましい結果にはなりません)。必ず「rake pdf」や「rake epub」を使ってください。

繰り返しになりますが、エラーが発生したときは「コンパイルエラーになったら」(p.19)の説明を読んでみてください。

第 2 章

チュートリアル

前の章では、Ruby と TeXLive のインストールを説明し、サンプル PDF ファイルを生成しました。

この章では、自分で書いた原稿ファイルから PDF ファイルを生成する方法を説明します。

なおこの章は、Ruby と TeXLive のインストールが済んでいること、またサンプルの PDF ファイルが生成できたことを前提にしています。まだの人は前の章を見てください。

2.1 用語の説明

このあとの説明で使用する用語を紹介します。

Starter

Re:VIEW Starter のことです。「Re:VIEW Starter」はちょっと長いので、この文章では省略して「Starter」と呼んでいます。

プロジェクトの zip ファイル

Re:VIEW Starter の Web サイトでプロジェクトを作ったときに、最後にダウンロードされる zip ファイルのことです。単に「zip ファイル」と呼ぶこともあります。

mybook.zip

説明で使用するプロジェクトの zip ファイル名です。これはサンプルであり、実際にはたとえば「samplebook.zip」だったり「brabra-book.zip」だったりします。

プロジェクトのフォルダ

zip ファイルを解凍したときにできるフォルダです。たとえば zip ファイル名が「mybook.zip」なら、解凍してできるフォルダは「mybook/」なので、これがプロジェクトのフォルダになります。

「mybook/」フォルダ

プロジェクトのフォルダのことです。フォルダ名はあくまでサンプルであり、プロジェクトごとに異なることに注意してください。

原稿ファイル

原稿となるテキストファイルです。これをもとに PDF ファイルが生成されます。原稿ファ

イルは拡張子が「.re」なので、「*.re ファイル」と呼ぶこともあります。なお文字コードは必ず UTF-8 にしてください。

***.re ファイル**

原稿ファイルのことです。拡張子が「.re」なので、こう呼ばれることがあります。

「contents/」フォルダ

原稿ファイルが置かれているフォルダです。最近の Re:VIEW Starter では、デフォルトで原稿ファイルを「contents/」フォルダに置くことになっています（以前はプロジェクトのフォルダ直下に置かれていました）。

テキストエディタ

テキストファイルを作成したり変更するためのアプリケーションのことです。単に「エディタ」と呼ぶこともあります。有名なものだと「Visual Studio Code」「Atom」「Emacs」「Vim」「サクラエディタ」「秀丸」「メモ帳」などがあります。

エディタ

テキストエディタのことです。

コンパイル

原稿ファイルから PDF ファイルを生成することです。Re:VIEW Starter では PDF だけでなく HTML や ePub のファイルも生成できるので、どの形式を生成するかを明示するなら「PDF へコンパイルする」「ePub へコンパイルする」などと言います。

2.2 フォルダとファイルの説明

プロジェクトの「mybook.zip」を解凍すると、数多くのファイルが作られます。それらのうち、重要なものを選んで解説します。すべてのファイルについての解説は付録 B「ファイルとフォルダ」を参照してください。

catalog.yml

原稿ファイルが順番に並べられたファイルです。原稿ファイルを追加した・削除した場合は、このファイルも編集します。

config-starter.yml

Starter 独自の設定ファイルです。Starter では「cofnig.yml」と「cofnig-starter.yml」の両方を使います。

config.yml

Re:VIEW の設定ファイルです。Starter によりいくつか変更と拡張がされています。

contents/

原稿ファイルを置くフォルダです*¹。

contents/*.re

原稿ファイルです。章 (Chapter) ごとにファイルが分かります。

images/

画像ファイルを置くフォルダです。この下に章 (Chapter) ごとのサブフォルダを作ることができます。

mybook-pdf/

PDF ファイルを生成するときの中間生成ファイルが置かれるフォルダです。I^AT_EX ファイルをデバッグするときに必要なとなりますが、通常は気にする必要はありません。

mybook.pdf

生成された PDF ファイルです。ファイル名はプロジェクトによって異なります。

review-ext.rb

Re:VIEW を拡張するためのファイルです。このファイルから「lib/ruby/*.rb」が読み込まれています。

sty/

I^AT_EX で使うスタイルファイルが置かれるフォルダです。

sty/mycolophon.sty

奥付*²の内容が書かれたスタイルファイルです。奥付を変更したい場合はこのファイルを編集します。

*¹ 原稿ファイルを置くフォルダ名は「config.yml」の「contentdir: contents」で変更できます。

*² 奥付とは、本のタイトルや著者や出版社や版や刷などの情報が書かれたページのことです。通常は本のいちばん最後のページに置かれます。

sty/mystyle.sty

ユーザが独自に L^AT_EX マクロを定義・上書きするためのファイルです。中身は空であり、ユーザが自由に追加して構いません。

sty/mytextsize.sty

PDF における本文の高さと幅を定義したファイルです。L^AT_EX では最初に本文の高さと幅を決める必要があるので、他のスタイルファイルから分離されてコンパイルの最初に読み込めるようになっています。

sty/mytitlepage.sty

大扉^{*3}の内容が書かれたスタイルファイルです。大扉のデザインを変更したい場合はこのファイルを編集します。

sty/starter*.sty

Starter 独自のスタイルファイルです。ここに書かれた L^AT_EX マクロを変更したい場合は、このファイルを変更するよりも「sty/mystyle.sty」に書いたほうがバージョンアップがしやすくなります。

^{*3} 大扉とは、タイトルページのことです。表紙のことではありません。

2.3 原稿の追加と変更

原稿の追加と変更の方法を説明します。

原稿の追加より変更のほうが簡単なので、変更する方法を先に説明します。

既存の原稿ファイルを変更する

プロジェクトのフォルダには、サンプルとなる原稿ファイルが存在します。まずはこれを変更してみましょう。

- (1) まず、お好みのテキストエディタを使って「mybook/contents/00-preface.re」を開いてください。
- (2) 次に、先頭の「= はじめに」の下に何か適当なテキストを追加して（例：リスト 2.1）、原稿ファイルを保存してください。
- (3) 原稿ファイルを保存したら、コンパイル^{*4}しましょう。Mac なら Terminal.app^{*5}、Windows ならコマンドプロンプトを開き、「rake pdf」（または Docker を使っているなら「rake docker:pdf」）を実行してください。

▼ リスト 2.1: 原稿ファイルの内容

= はじめに

見る、人がゴミのようだ！

←追加

...（以下省略）...

これで新しい PDF ファイルが生成されるはずです。生成された新しい PDF ファイルを開き、さきほど追加した行が「はじめに」の章に表示されていることを確認してください。

プロジェクトのフォルダに原稿ファイルがあるとコンパイルエラー

Starter のデフォルトでは、原稿ファイル (*.re) を「contents/」フォルダに置くよう設定されています。この場合、もし原稿ファイルがプロジェクトのフォルダ（たとえば「mybook/」の直下）にあると、コンパイル^{*6}できずエラーになります。

エラーになるのは、Starter の仕組みに起因します。Starter ではコンパイル時に、「contents/」に置かれた原稿ファイル (*.re) をプロジェクトフォルダの直下にコピーしてからコンパイルします^{*7}。そのため、プロジェクトフォルダ直下に別の原稿ファイルがあるとコンパイル時に上書きしてしまう可能性があるため、あえてエラーにしているのです。

^{*4} 「コンパイル」とは、ここでは原稿ファイルから PDF ファイル（または HTML や ePub）を生成することです。このあとも使う用語なので覚えてください。また用語については「2.1 用語の説明」（p.11）も参照してください。

^{*5} Terminal.app は、ファインダーで「アプリケーション」フォルダ > 「ユーティリティ」フォルダ > 「ターミナル」をダブルクリックすると起動できます。

^{*6} コンパイルとは、ここでは原稿ファイルから PDF ファイルを生成することを指します。

^{*7} Re:VIEW 2.5 の仕様上、コンパイル時に原稿ファイルがプロジェクトフォルダの直下にないとけないため。

.....

ここまでが、既存の原稿ファイルを変更する方法でした。次に、新しい原稿ファイルを追加する方法を説明します。

新しい原稿ファイルを追加する

新しい原稿ファイルを追加するには、次のようにします。

- (1) 新しい原稿ファイルを作成する。
- (2) 作成した原稿ファイルをプロジェクトに登録する。
- (3) PDF ファイルを生成し、原稿の内容が反映されることを確認する。

順番に説明しましょう。

(1) 新しい原稿ファイルを作成する

お好みのテキストエディタを使って、新しい原稿ファイル「01-test.re」を作成します。

このとき、次の点に注意してください。

- 原稿ファイルの拡張子は「.re」にします。
- 原稿ファイルの置き場所は「contents/」フォルダにします。つまりファイルパスは「mybook/contents/01-test.re」になります。
- 原稿ファイルの文字コードは「UTF-8」にします。

新しい原稿ファイル「contents/01-test.re」の中身は、たとえばリスト 2.2 のようにしてください。

▼ リスト 2.2: 原稿ファイル「contents/01-test.re」

```
= サンプル章タイトル

//abstract{
概要概要概要
//}

== サンプル節タイトル
本文本文本文
```

.....

原稿ファイルに番号をつけるべきか

この例では原稿ファイル名を「01-test.re」にしました。しかしファイル名に「01-」のような番号は、必ずしもつける必要はありません（つけてもいいし、つけなくてもいい）。

ファイル名に番号をつけるのは、利点と欠点があります。

- 利点は、Mac のファインダや Windows のエクスプローラで表示したときに、ファイルが章の順番に並ぶことです。

- 欠点は、章の順番を入れ替えるときにファイル名の変更が必要なこと、またそれにより章 ID^{*8}が変わるのでその章や節を参照している箇所もすべて変更になることです。番号をつけていない場合は、「catalog.yml」での順番を入れ替えるだけで済み、参照箇所も修正する必要はありません。

自分の好きなほうを選んでください。

.....

(2) 原稿ファイルをプロジェクトに登録する

次に、作成した原稿ファイルをプロジェクトに登録しましょう。プロジェクトのフォルダにある「catalog.yml」をテキストエディタで開き、リスト 2.3 のように変更してください。なお、このファイルでは「#」以降は行コメントであり読み飛ばされます。

▼ リスト 2.3: ファイル「catalog.yml」

```
PREDEF:
- 00-preface.re

CHAPS:
- 01-install.re    ←この行を削除
- 01-test.re         ←この行を追加
- 02-tutorial.re

APPENDIX:

POSTDEF:
- 99-postface.re

##
## ■Tips
## ... (以下省略) ...
```

ファイルを変更したら、忘れずに保存してください。

これで、新しい原稿ファイルがプロジェクトに登録できました。

(3) PDF ファイルを生成し、原稿の内容が反映されたか確認する

試しに PDF ファイルを生成してみましょう。Mac なら Terminal.app、Windows ならコマンドプロンプトを開き、次のコマンドを実行してください。

▼ PDF ファイルを生成する

```
$ rake pdf          ← Docker を使っていない場合
$ rake docker:pdf   ← Docker を使っている場合
```

^{*8} 章 ID については「章 ID」(p.48) で説明します。

新しい PDF ファイルが生成されるので、表示してみてください。新しい原稿ファイルの章が追加されていれば成功です。

「catalog.yml」の説明

「catalog.yml」の内容は次のような構造になっています。

- 「PREDEF:」は「まえがき」を表します。
- 「CHAPS:」は本文を表します。
- 「APPENDIX:」は付録を表します。
- 「POSTDEF:」は「あとがき」を表します。

▼ ファイル「catalog.yml」

```
PREDEF:
- 00-preface.re      ←まえがきの章

CHAPS:
- 01-install.re      ←本文の章
- 02-tutorial.re     ←本文の章
- 03-syntax.re       ←本文の章

APPENDIX:
- 92-filelist.re     ←付録の章

POSTDEF:
- 99-postfix.re      ←あとがきの章
```

また次のような点に注意してください。

- まえがきや付録やあとがきは、なければ省略できます。
- まえがきとあとがきには、章番号や節番号がつきません。
- 目次はまえがきのあとに自動的につけられます*⁹。
- 大扉（タイトルページ）や奥付も自動的につけられます*¹⁰。

.....

YAML 形式

「catalog.yml」の内容は、「YAML」という形式で書かれています。YAML については Google 検索で調べてください。ここでは 3 点だけ注意点を紹介します。

- タブ文字を使わないこと。
- 「-」のあとに半角空白をつけること。
- 「#」以降は行コメントとして読み飛ばされる。

*⁹ 目次をつけたくない場合は、「config.yml」に「toc: false」を設定してください。

*¹⁰ 大扉をつけたくない場合は、「config.yml」に「titlepage: false」を設定してください。また奥付をつけたくない場合は「colophon: false」を設定してください。

.....

コンパイルエラーになったら

PDF ファイルの生成がエラーになったら、以下の点を確認してください。

- インライン命令がきちんと閉じているか
- ブロック命令の引数が足りているか、多すぎないか
- 「//」が足りてないか、または多すぎないか
- 「@<fn>{ }」や「@{ }」や「@<table>{ }」のラベルが合っているか
- 「@<chapref>{ }」で指定した章 ID が合っているか
- 「@<secref>{ }」で指定した節や項が存在するか
- 脚注の中で「】」を「\】」とエスケープしているか
- 「//image」で指定した画像ファイルが存在するか
- 原稿ファイル名を間違っていないか
- 原稿ファイルの文字コードが UTF-8 になっているか
- 索引に何も登録してないのに索引機能をオンにしていないか（詳細は「3.17 用語、索引」(p.121) を参照)

詳しくは「5.1 コンパイルエラー」(p.165) を見てください。

それでもエラーになる場合は、中間ファイルを削除してみましょ。たとえばプロジェクト名が「mybook」だったら、「mybook-pdf」というフォルダが作られているはずです。これを削除してから、コンパイルし直してみてください。

▼ 中間ファイルを削除してコンパイルし直す

```
$ rm -rf mybook-pdf      ←中間ファイルを削除して、
$ rake pdf                ←コンパイルし直す
```

それでもダメなら、「#reviewstarter」というタグでツイートしていただければ、相談に乗ります。

文章をチェックする

Starter では、「textlint」というツールを使って原稿の文章をチェックできます。

たとえば「Starter を使うとききれいな PDF を作成することができます」という文章は、「Starter を使うとききれいな PDF を作成できます」のように簡潔にできます。このような指摘を行ってくれるのが textlint です。

textlint を Starter のプロジェクトで使う方法は、「6.11 textlint による文章チェック」(p.212) で解説しています。今はまだ textlint を使う必要はありませんが、Starter に慣れてきたら使ってみてください。

2.4 基本的な記法

原稿ファイルは、ある決まった書き方（記法）に従って記述します。たとえば、次のような記法があります。

- 章 (Chapter) は「= 」で始め、節 (Section) は「== 」で始める。
- 箇条書きは「 * 」で始める。
- プログラムコードは「`//list{...//}`」で囲う。
- 強調は「`@{...}`」または「`@{...}`」で囲う。

ここでは記法のうち基本的なものを説明します。詳しいことは第3章「記法」で説明します。

コメント

行コメントは「`#@#`」で、また範囲コメントは「`#@+++`」と「`#@---`」で表します。どちらも行の先頭から始まってないと、コメントとして認識されません。

▼ サンプル

```
本文1。
#@#本文2。
本文3。

#@+++
本文4。
#@---
本文5。
```

▼ 表示結果

```
本文 1。本文 3。
本文 5。
```

詳しくは「3.1 コメント」(p.39)を参照してください。

段落と改行

空行があると、新しい段落になります。改行は無視されるか、または半角空白扱いになります。通常は1文ごとに改行して書きます。

▼ サンプル

```
言葉を慎みたまえ。君はラピュタ王の前にいるのだ。

これから王国の復活を祝って、諸君にラピュタの力を見せてやろうと思ってね。
見せてあげよう、ラピュタの雷を！
```

旧約聖書にあるソドムとゴモラを滅ぼした天の火だよ。
ラーマヤーナではインドラの矢とも伝えているがね。

▼ 表示結果

言葉を慎みたまえ。君はラピュタ王の前にいるのだ。
これから王国の復活を祝って、諸君にラピュタの力を見せてやろうと思ってね。見せてあげよう、ラピュタの雷を！
旧約聖書にあるソドムとゴモラを滅ぼした天の火だよ。ラーマヤーナではインドラの矢とも伝えているがね。

詳しくは「3.2 段落と改行と空行」(p.41)を参照してください。

見出し

章 (Chapter) や節 (Section) といった見出しは、「= 」や「== 」で始めます。また章には概要を書くといいでしょう。

▼ サンプル

```
= 章(Chapter)見出し

//abstract{
章の概要。
//}

== 節(Section)見出し

=== 項(Subsection)見出し

==== 目(Subsubsection)見出し

===== 段(Paragraph)見出し

===== 小段(Subparagraph)見出し
```

Starter では、章 (Chapter) は 1 ファイルにつき 1 つだけにしてください。1 つのファイルに複数の章 (Chapter) を入れないでください。

見出しについての詳細は「3.3 見出し」(p.46)を参照してください。

箇条書き

番号なし箇条書きは「 * 」で始めます（先頭に半角空白が必要）。

▼ サンプル

```
* 箇条書き1
* 箇条書き2
** 入れ子の箇条書き
*** 入れ子の入れ子
```

▼ 表示結果

- 箇条書き 1
 - 箇条書き 2
 - 入れ子の箇条書き
 - * 入れ子の入れ子
-

番号つき箇条書きは「 1. 」のように始める方法と、「 - A. 」のように始める方法があります（どちらも先頭に半角空白が必要）。後者では番号の自動採番はされず、指定された文字列がそのまま出力されます。

▼ サンプル

```
1. この記法では数字しか指定できず、
2. また入れ子にもできない。

- (A) この記法だと、英数字だけでなく
- (B) 任意の文字列が使える
-- (B-1) 入れ子もできるし、
*** 番号なし箇条書きも含められる
```

▼ 表示結果

- 1. この記法では数字しか指定できず、
 - 2. また入れ子にもできない。
 - (A) この記法だと、英数字だけでなく
 - (B) 任意の文字列が使える
 - (B-1) 入れ子もできるし、
 - 番号なし箇条書きも含められる
-

箇条書きの詳細は「3.4 箇条書きと定義リスト」(p.54) を参照してください。

用語リスト

用語リスト（HTML でいうところの「<dl><dt><dd>」）は、「：」で始めて^{*11}、次の行からインデントします^{*12}。

▼ サンプル

```
： 用語1
  説明文。
  説明文。
： 用語2
  説明文。
  説明文。
```

▼ 表示結果

用語 1

説明文。説明文。

用語 2

説明文。説明文。

詳しくは「定義リスト」(p.55) を参照してください。

太字と強調

太字は「@{...}」で囲み、強調は「@{...}」または「@{...}」で囲みます。強調は、太字になるだけでなくフォントがゴシック体になります。

▼ サンプル

```
テキスト@<b>{テキスト}テキスト。
```

```
テキスト@<B>{テキスト}テキスト。
```

▼ 表示結果

テキスト**テキスト**テキスト。

テキスト**テキスト**テキスト。

日本語の文章では、強調するときは太字のゴシック体にするのがよいとされています。なので強調には「@{...}」または「@{...}」を使い、「@{...}」は使わないでください。

^{*11} 先頭の半角空白は入れるようにしましょう。過去との互換性のため、先頭の半角空白がなくても動作しますが、箇条書きとの整合性のために半角空白を入れることを勧めます。

^{*12} 説明文のインデントは、半角空白でもタブ文字でもいいです。

強調や太字などテキストの装飾についての詳細は「3.6 強調と装飾」(p.63)を参照してください。

インライン命令

「@{...}」のような記法はインライン命令と呼ばれます。インライン命令は入れ子にできますが (Starter による拡張)、複数行を含めることはできません。詳細は「インライン命令」(p.61)を見てください。

等幅フォント

文字列を等幅フォントで表示するには、「@<tt>{...}」を使います。ただしこれを使うことはお勧めしません。それよりも、それぞれの用途に合ったインライン命令を使いましょう。

- プログラムコードやコマンド文字列を表すには「@<code>{...}」を使ってください。違いの分かりやすい例でいうと、たとえば「@<tt>{''}`」は「''」と表示され、「@<code>{''}`」は「`」と表示されます*13。
- ファイル名を表すには「@<file>{...}」を使ってください。等幅フォントではなくゴシック体で表示されます。

プログラムリスト

プログラムリストは「//list[ラベル][説明文]{ ... //}」で囲みます。

- ラベルは、他と重複しない文字列にします。
- 「@<list>{ラベル名}」のようにプログラムリストを参照できます。
- 説明文に「]」を含める場合は、「\]」のようにエスケープします。

▼ サンプル

サンプルコードを@<list>{fib1}に示します。

```
//list[fib1][フィボナッチ数列]{
def fib(n):
    return n if n <= 1 else fib(n-1) + fib(n-2)
//}
```

▼ 表示結果

サンプルコードをリスト 2.4 に示します。

▼ リスト 2.4: フィボナッチ数列

*13 使用する等幅フォントによっては同じように表示されます。


```
def fib(n):
    return n if n <= 1 else fib(n-1) + fib(n-2)
```

第 1 引数と第 2 引数のどちらも、省略できます。たとえば第 1 引数だけを省略するには「`//list[]`」[説明]「`{}`」のようにします。両方とも省略するには「`//list[]`」[説明]「`{}`」または「`//list`」[説明]「`{}`」のようにします。

またプログラムリストの中で、インライン命令が使えます^{*14}。たとえば次の例では、追加した箇所を「`@<ins>{...}`」で、削除した箇所を「`@{...}`」で表しています。

▼ サンプル

```
//list{
function fib(n) {
    @<del>|if (n <= 1) { return n; }|
    @<del>|else          { return fib(n-1) + fib(n-2); }|
    @<ins>|return n <= 1 ? n : fib(n-1) + fib(n-2);|
}
//}
```

▼ 表示結果

```
function fib(n) {
    if (n <= 1) { return n; }
    else { return fib(n-1) + fib(n-2); }
    return n <= 1 ? n : fib(n-1) + fib(n-2);
}
```

その他、行番号をつけたり外部ファイルを読み込んだりできます。プログラムコードについての詳細は「3.7 プログラムリスト」(p.66)を参照してください。

また出力結果を表す「`//output`」という命令もあります。詳細は「出力結果」(p.73)を参照してください。

.....

ブロック命令

「`//list{ ... //}`」のような形式の命令を、**ブロック命令**といいます。ブロック命令は入れ子にできますが（Starter による拡張）、できないものもあります。詳細は「ブロック命令」(p.62)を見てください。

.....

^{*14} プログラムリストの中では、「`@{}`」（強調）ではなく「`@{}`」（太字）を使ってください。なぜなら、「`@{}`」を使うとフォントが等幅フォントからゴシック体に変更されてしまい、表示がずれてしまうからです。

ターミナル画面

ターミナル（端末）の画面は、「`//terminal{ ... //}`」を使います。次の例では、引数にラベル名と説明文字列を指定します。また「`@<userinput>{...}`」はユーザ入力を表します。

▼ サンプル

```
//terminal[term-1][PDFを生成]{
$ @<userinput>{rake pdf}          @<balloon>{Dockerを使わない場合}
$ @<userinput>{rake docker:pdf}    @<balloon>{Dockerを使う場合}
//}
```

▼ 表示結果

▼ リスト 2.5: PDF を生成

```
$ rake pdf          ← Dockerを使わない場合
$ rake docker:pdf   ← Dockerを使う場合
```

ラベル名を指定していれば、プログラムリストと同じように「`@<list>{ラベル名}`」で参照できます。またラベル名と説明文字列はどちらも省略可能です。どちらも省略すると「`//terminal{ ... //}`」のように書けます。

詳しくは「3.9 ターミナル」(p.79)を参照してください。

脚注

脚注は、脚注をつけたい箇所に「`@<fn>{ラベル}`」をつけ、段落の終わりに「`//footnote[ラベル][脚注文]`」を書きます。脚注文の中では「`]`」を「`\]`」と書いてください。

▼ サンプル

```
本文テキスト@<fn>{fn-12}。

//footnote[fn-12][このように脚注文を書きます。]
```

▼ 表示結果

本文テキスト*15。

図

図を入れるには、「`//image[画像ファイル名][説明文][オプション]`」を使います。

- 画像ファイルは「images/」フォルダに置きます。

*15 このように脚注文を書きます。

- 画像ファイルの拡張子は指定しません。
- ラベルを使って「@{ラベル}」のように参照できます。

▼ サンプル

```
//image[tw-icon][サンプル画像][scale=0.3]
```

▼ 表示結果



▲ 図 2.1: サンプル画像

Starter では、画像を入れる位置を指定したり、画像の周りに枠線をつけたりできます。詳しくは「3.13 画像」(p.91) を参照してください。

ノート

Starter では、本文を補足する文章を「ノート」という囲み枠で表します。

▼ サンプル

```
//note[ムスカの苦手なもの]{
実は、ムスカには「虫が苦手」という公式な設定があります。
有名な『読める、読めるぞ!』のシーンでムスカが顔の周りに群がるハエを追い払うのは、邪
魔だったからだけでなく、虫が苦手だったからです。
//}
```

▼ 表示結果

.....
ムスカの苦手なもの

実は、ムスカには「虫が苦手」という公式な設定があります。有名な『読める、読めるぞ!』のシーンでムスカが顔の周りに群がるハエを追い払うのは、邪魔だったからだけでなく、虫が苦手だったからです。

.....

ノート本文には簡条書きやプログラムコードを埋め込めます。詳しくは「3.11 ノート」(p.83) を参照してください。

また、補足情報や注意喚起や警告を表す囲み枠もあります。詳しくは「ノート以外のミニブロック」(p.85)を参照してください。

▼ サンプル

```
//info[解決のヒント]{  
本製品を手に取り、古くから伝わるおまじないを唱えてみましょう。  
すると天空の城への門が開きます。  
//}  
  
//caution[使用上の注意]{  
本製品を石版にかざすと、天空の城から雷が発射されます。  
周りに人がいないことを確かめてから使用してください。  
//}  
  
//warning[重大な警告]{  
本製品を持ったまま滅びの呪文を唱えると、天空の城は自動的に崩壊します。  
大変危険ですので、決して唱えないでください。  
//}
```

▼ 表示結果



解決のヒント

本製品を手に取り、古くから伝わるおまじないを唱えてみましょう。すると天空の城への門が開きます。



使用上の注意

本製品を石版にかざすと、天空の城から雷が発射されます。周りに人がいないことを確かめてから使用してください。



重大な警告

本製品を持ったまま滅びの呪文を唱えると、天空の城は自動的に崩壊します。大変危険ですので、決して唱えないでください。

表

表は、次のように書きます。

- 「`//table[ラベル][説明文]{ ... //}`」で囲みます。
- セルは1つ以上のタブで区切ります。
- ヘッダの区切りは「-」または「=」を12個以上並べます。
- ラベルを使って「`@<table>{ラベル}`」のように参照できます。

▼ サンプル

```
//table[tbl-31][サンプル表]{
Name      Val1      Val2
-----
AA         12        34
BB         56        78
//}
```

▼ 表示結果

▼ 表 2.1: サンプル表

Name	Val1	Val2
AA	12	34
BB	56	78

PDF ではセルの右寄せ・左寄せ・中央揃えができます。また CSV ファイルからデータを読み込めます。詳しくは「3.14 表」(p.98) を参照してください。

数式

LaTeX の書き方を使って数式を記述できます。

▼ サンプル

```
//texequation[euler][オイラーの公式]{
e^{i\theta} = \sin{\theta} + i\cos{\theta}
//}
```

@<m>\$\theta = \pi\$のときは@<m>\$e^{i\pi} = -1\$となり、これはオイラーの等式と呼ばれます。

▼ 表示結果

式 2.1: オイラーの公式

$$e^{i\theta} = \sin \theta + i \cos \theta$$

$\theta = \pi$ のときは $e^{i\pi} = -1$ となり、これはオイラーの等式と呼ばれます。

詳しくは「3.15 数式」(p.112) を参照してください。

会話形式

Starter では、会話形式も書けます。会話形式は、アイコン画像を使う方法と使わない方法があります。

▼ サンプル

```
//talklist{
//talk[avatar-b]{
あの方は何も盗らなかったわ。私のために闘ってくださったんです。
//}
//talk[avatar-g]{
いや、ヤツはとんでもないものを盗んでいきました……
あなたの心です。
//}
//}
```

▼ 表示結果



「あの方は何も盗らなかったわ。私のために闘ってくださったんです。」



「いや、ヤツはとんでもないものを盗んでいきました……あなたの心です。」

▼ サンプル

```
//talklist{
//talk[][クラリス]{
あの方は何も盗らなかったわ。私のために闘ってくださったんです。
//}
//talk[][銭形警部]{
いや、ヤツはとんでもないものを盗んでいきました……
```

あなたの心です。

//}

//}

▼ 表示結果

クラリス：「あの方は何も盗らなかったわ。私のために闘ってくださったんです。」

銭形警部：「いや、ヤツはとんでもないものを盗んでいきました……あなたの心です。」

詳しくは「3.16 会話形式」(p.114)を参照してください。

白黒印刷でも判別できる画像を使う

このサンプルでは、色が違うだけのアイコン画像を使っています。そのせいで、白黒印刷すると誰が誰だか判別できなくなります。もし白黒印刷するなら、色に頼らず判別できるようなアイコン画像を使いましょう。

会話形式にしても分かりやすくはならない

会話形式を採用しても、説明が分かりやすくなるとは限りません。会話形式にすると楽しさや親しみやすさを向上できますが、それは説明の分かりやすさとは違います。

会話形式だけど説明がよく分からない本はたくさんあります。先生役や先輩役のキャラクターが何かの説明をして、それが読者には伝わっていないのに、生徒役や後輩役のキャラクターが「なるほど!」とか「そういうことか!」と言っているような本、見たことはありませんか? 先生や先輩の説明を苦もなく理解できるほどの優秀な生徒や後輩は、本の中にしかいません。

同様のことはマンガ形式にもいえます。分かりやすく説明したいなら分かりやすく説明する技術を学ぶべきであり、マンガにしたり会話形式にすることではありません。

たとえば、『わかばちゃんと学ぶ Git 使い方入門^{*16}』が分かりやすいのはマンガだからではなく、分かりやすく説明する技術を作者の湊川先生が身につけているからです^{*17}。分かりやすく説明できる人がたまたまマンガで説明しているから分かりやすいのであって、マンガや会話形式にすれば何でも分かりやすくなるわけではありません。ここを履き違えないようにしましょう。

用語、索引

用語は「@<term>{用語名 ((よみかた))}」のように書くとゴシック体で表示され、かつ本の巻末の索引に載ります。ゴシック体にするけど索引に載せないなら「@<termnoidx>{用語名}」、ゴ

^{*16} <https://www.c-r.com/book/detail/1108>

^{*17} 湊川先生による文章改善の発表動画 (<https://www.youtube.com/watch?v=t2eiA2ylNW0>) を見ると、分かりやすく説明する技術を先生自身が持っていることがよく分かります。

シック体にせず索引に載せるなら「@<idx>{用語名 ((よみかた))}」、索引に載せるけど表示はしないなら「@<hidx>{用語名 ((よみかた))}」とします。

索引は、デフォルトではオフになっています。索引を使うには、`config.yml`で「`makeindex: true`」を設定してください (370 行目ぐらいにあります)。索引は \LaTeX の機能を使うので、今のところ PDF でしかサポートされません。また索引機能をオンにすると \LaTeX のコンパイル回数が増えるので、コンパイル時間が延びます。原稿執筆中はオフにし、完成間近になったらオンにするといいでしょう。

なお索引に登録する用語が何もない場合は、索引を使おうとするとコンパイルエラーになります。本文に1つでも「@<term>{ }」や「@<idx>{ }」を入れてから索引機能を有効化してください。

Starter では索引語の親子関係を指定したり、ページ番号のかわりに「～を見よ」と指定できます。詳しくは「3.17 用語、索引」(p.121) を参照してください。

単語展開

キーを単語に展開できます。たとえば辞書ファイル「`data/words.txt`」の中に、キー「apple」に対応した単語「アップル」が登録されているとします。すると、本文に「@<w>{apple}」と書くと「アップル」に展開されます。「@<wb>{apple}」なら展開して太字になります。また「@<W>{apple}」なら展開して強調表示します。

この機能は、表記が定まっていない単語に使うと便利です。たとえば「Apple」と「アップル」のどちらの表記にするか悩んでるなら、本文には「アップル」と書いておけば、あとから辞書の中身を変えるだけでどちらの表記にも対応できます。

詳しくは「3.18 単語展開」(p.128) を参照してください。

2.5 印刷用 PDF と電子用 PDF

Starter では、印刷用と電子用とを切り替えて PDF ファイルを生成できます。両者の違いはあとで説明するとして、まず印刷用と電子用の PDF ファイルを生成する方法を説明します。

印刷用と電子用を切り替えて PDF を生成する

Starter では、印刷用と電子用とを切り替えて PDF を生成できます（デフォルトは印刷用）。具体的には次のどれかを行います^{*18}。

- 「`rake pdf t=pbook`」を実行すると印刷用 PDF が生成され、「`rake pdf t=ebook`」^{*19}を実行すると電子用 PDF が生成される。
- 環境変数「`$STARTER_TARGET`」を「`pbook`」に設定してから「`rake pdf`」を実行すると印刷用 PDF が生成され、「`ebook`」に設定してから実行すると電子用 PDF が生成される。
- 「`config-starter.yml`」に設定項目「`target:`」があるので、これを「`pbook`」（印刷用）または「`ebook`」（電子用）に設定する。

▼ macOS や Linux の場合^{*20}

```
$ rake pdf t=pbook      ←印刷用PDFを生成
$ rake pdf t=ebook      ←電子用PDFを生成
## または
$ export STARTER_TARGET=pbook ←印刷用に設定
$ rake pdf              ←またはDockerを使うなら rake docker:pdf
$ export STARTER_TARGET=ebook ←電子用に設定
$ rake pdf              ←またはDockerを使うなら rake docker:pdf
```

Windows の場合は、「`set STARTER_TARGET=pbook`」や「`set STARTER_TARGET=ebook`」を使って環境変数を設定してください。

環境変数を未設定に戻すには、次のようにします。

▼ macOS や Linux の場合

```
$ unset STARTER_TARGET ←環境変数を未設定に戻す
```

印刷用 PDF と電子用 PDF の違い

印刷用 PDF と電子用 PDF には、次のような違いがあります。

カラー

印刷用では、カラーは使われず白黒になります（画像は除く）。

^{*18} 強制力はこの順序であり、設定ファイルより環境変数のほうが、また環境変数より「`rake`」コマンドのオプションのほうが優先されます。

^{*19} 「`pbook`」は「`printing book`」、「`ebook`」は「`electronic book`」を表します。

電子用では、カラーが使われます*²¹。

左右の余白幅

印刷用では、左右の余白幅が異なります。具体的には、見開きにおいて内側の余白を約 2cm、外側の余白を約 1.5cm にしています*²²。これは見開きでの読みやすさを確保したうえで本文幅をできるだけ広げるための工夫です。

電子用では見開きを考慮する必要がないので、左右の余白幅は同じに設定されます。

ノンブル

印刷用には自動的にノンブルがつき、電子用にはつきません。「ノンブル」とはすべてのページにつけられる通し番号であり、印刷所に入稿するときが必要です。ノンブルについての詳細は「5.5 ノンブル」(p.174)を参照してください。

表紙

印刷用では、表紙がつきません。なぜなら、表紙の PDF ファイルは本文とは別にして印刷所に入稿するからです。

電子用では、(設定されていれば)表紙がつきます*²³。

*²¹ カラーの設定は「sty/starter-color.sty」を見てください。変更する場合はこのファイルではなく「sty/mystyle.sty」に書くといいでしょう。

*²² 余白幅は初期設定によって多少の違いがあります。設定の詳細は「sty/mytextsize.sty」を見てください。

*²³ 表紙のつけ方は「[PDF] 表紙や大扉や奥付となる PDF ファイルを指定する」(p.157)を見てください。

2.6 高速プレビュー

ページ数が多くなると、PDF ファイルへのコンパイルに時間がかかるようになり、執筆に支障が出ます。

ここでは高速にプレビューするための機能を紹介します。

指定した章だけをコンパイルする

Starter では、指定した章 (Chapter) だけを L^AT_EX でコンパイルできます。これは章の数が多い場合や、著者が多数いる合同誌の場合にはとても効果的です。具体的には次のようにします。

- 「`rake pdf c=03-syntax`」のようにすると、「03-syntax.re」だけをコンパイルします。
- または環境変数「`$STARTER_CHAPTER`」を設定する方法でも、章を指定できます。

▼例：03-syntax-faq.re だけをコンパイルする

```
$ rake pdf c=03-syntax ←「.re」はつけない
### または
$ export STARTER_CHAPTER=03-syntax ←「.re」はつけない
$ rake pdf ← Dockerを使っているなら rake docker:pdf
```

コンパイルする章を指定したとき、Starter は次のような動作になります。

- 他の章は無視されます。
- 表紙や、目次や、大扉や、奥付も無視されます。
- L^AT_EX のコンパイル回数が 1 回だけになります (コンパイル時間を短縮するため)。

なお「`$STARTER_CHAPTER`」を設定した場合、全体をコンパイルするときにはリセットしてください。

▼全体をコンパイルする

```
$ unset STARTER_CHAPTER ←「$」はつけないことに注意
```

コンパイル回数を 1 回だけに制限する

Re:VIEW および Starter では、L^AT_EX のコンパイルが最大で 3 回行われます^{*24}。

- 1 回目のコンパイルで章や節のページ番号が分かる。
- 2 回目のコンパイルで目次が作成される。
- もし 2 回目のコンパイルでページ番号が変わると、3 回目のコンパイルが行われる。

しかし、ページ番号が合ってなくてもいいからとにかく仕上がりを確認したい場面はよくあり、そんなときは 3 回もコンパイルされるのは時間の無駄です。

^{*24} 索引があると、コンパイル回数ももう 1 回増えます。

そこで Starter では、 \LaTeX のコンパイル回数を制限する機能を用意しました。

- 「`rake pdf n=1`」のようにすると、 \LaTeX のコンパイルが 1 回しか行われません。
- または環境変数「`$STARTER_COMPILETIMES`」を「1」に設定する方法でも、同じように制限できます。

▼ \LaTeX のコンパイル回数を 1 回に制限してコンパイル

```
$ rake pdf n=1
### または
$ STARTER_COMPILETIMES=1 rake pdf
```

画像読み込みを省略するドラフトモード

Starter では、画像の読み込みを省略する「ドラフトモード」を用意しました。ドラフトモードにすると、画像のかわりに枠線が表示されます。こうすると、(\LaTeX のコンパイル時間は変わりませんが) DVI ファイルから PDF を生成する時間が短縮されます。

図やスクリーンショットが多い場合や、印刷用に高解像度の画像を使っている場合は、この機能は特に効果が高いです。

ドラフトモードにするには、次のどれを実行します*25。

- 「`rake pdf d=on`」でコンパイルする。
- 環境変数「`$STARTER_DRAFT`」の値を「on」に設定してからコンパイルする。
- `config-starter.yml` で「`draft: true`」を設定してからコンパイルする。

▼ 画像読み込みを省略するドラフトモードで PDF を生成する

```
$ rake pdf d=on      ←ドラフトモードをonにする
### または
$ export STARTER_DRAFT=on ←ドラフトモードをonにする
$ rake pdf           ←またはDocker環境なら rake docker:pdf
```

環境変数の設定をクリアするには、「`unset STARTER_DRAFT`」を実行してください。

また「ドラフトモードにして PDF 生成時間を短縮したい、でもこの画像は表示して確認したい」というときもあるでしょう。そんなときは「`//image[...][...]draft=off`」のように指定すると、その画像はドラフトモードが解除されて PDF に表示されます。

自動リロードつき HTML プレビュー

Starter では、HTML でプレビューするための機能を用意しました。便利なことに、原稿を変更すると自動的にリロードされます。PDF と比べて HTML の生成はずっと高速なので、原稿執筆中に入力間違いを見つけるには HTML のほうが向いています。

*25 強制力はこの順序であり、設定ファイルより環境変数のほうが、また環境変数より「`rake`」コマンドのオプションのほうが優先されます。

使い方は、まず Web サーバを起動します。

▼ Web サーバを起動する

```
$ rake web:server      ← Dockerを使っていない場合
$ rake docker:web:server ← Dockerを使っている場合
```

起動したらブラウザで <http://localhost:9000/> にアクセスし、適当な章を開いてください。そして開いた章の原稿ファイル (*.re) を変更すると、ブラウザの画面が自動的にリロードされ、変更が反映されます。

原稿執筆中は、エディタのウィンドウの後ろにプレビュー画面が少し見えるようにするといいでしょう。

いくつか注意点があります。

- 表示は HTMLで行っているため、PDFでの表示とは差異があります。執筆中は HTMLでプレビューし、区切りのいいところで PDFによる表示を確認するといいでしょう。
- 今のところ数式はプレビューできません。
- 変更が反映されるのは、開いているページと対応した原稿ファイルが変更された場合だけです。たとえば「foo.html」を開いているときに「foo.re」を変更するとプレビューに反映されますが、別の「bar.re」を変更しても反映されません。
- 画面右上の「Rebuild and Reload」リンクをクリックすると、原稿ファイルが変更されていなくても強制的にコンパイルとリロードがされます。
- 原稿ファイルに入力間違いがあった場合は、画面にエラーが表示されます。エラー表示はあまり分かりやすくないので、今後改善される予定です。
- Web サーバを終了するには、Control キーを押しながら「c」を押してください。

第 3 章

記法

この章では、原稿ファイルの記法について詳しく説明します。初めての人は、先に「2.4 基本的な記法」(p.20)を見たほうがいいでしょう。

3.1 コメント

行コメント

「#@#」で始まる行は行コメントであり、コンパイル時に読み飛ばされます。一般的な行コメントとは違って、行の先頭から始まる必要があるので注意してください。

▼ サンプル

```
本文1。  
#@#本文2。  
本文3。
```

▼ 表示結果

本文 1。本文 3。

..... 行コメントを空行扱いにしない

Re:VIEW では行コメントを空行として扱います。たとえば上の例の場合、「本文 1」と「本文 3」が別の段落になってしまいます。ひどい仕様ですね。こんな仕様だと、段落の途中をコメントアウトするときにとっても不便です。

Starter では「本文 1」と「本文 3」が別の段落にならないよう仕様を変更しています。

.....

範囲コメント

「#@+++」から「#@---」までの行は範囲コメントであり、コンパイル時に読み飛ばされます。

▼ サンプル

本文1。

#@+++

本文2。

本文3。

#@---

本文4。

▼ 表示結果

本文 1。

本文 4。

範囲コメントは入れ子にできません。また「+」「-」の数は3つと決め打ちされています。
なお範囲コメントは Starter による拡張機能です。

3.2 段落と改行と空行

段落

空行を入れると、段落の区切りになります。空行は何行入れても、1 行の空行と同じ扱いになります。

▼ サンプル

言葉を慎みたまえ。君はラピュタ王の前にいるのだ。

これから王国の復活を祝って、諸君にラピュタの力を見せてやろうと思ってね。見せてあげよう、ラピュタの雷を！

旧約聖書にあるソドムとゴモラを滅ぼした天の火だよ。ラーマヤーナではインドラの矢とも伝えてるがね。

▼ 表示結果

言葉を慎みたまえ。君はラピュタ王の前にいるのだ。

これから王国の復活を祝って、諸君にラピュタの力を見せてやろうと思ってね。見せてあげよう、ラピュタの雷を！

旧約聖書にあるソドムとゴモラを滅ぼした天の火だよ。ラーマヤーナではインドラの矢とも伝えてるがね。

段落は、PDF なら先頭の行を字下げ（インデント）し、ePub なら 1 行空けて表示されます。また段落の前に「//noindent」をつけると、段落の先頭の字下げ（インデント）をしなくなります。

▼ サンプル

//noindent

言葉を慎みたまえ！君はラピュタ王の前にいるのだ！

//noindent

これから王国の復活を祝って、諸君にラピュタの力を見せてやろうと思ってね。見せてあげよう、ラピュタの雷を！

//noindent

旧約聖書にあるソドムとゴモラを滅ぼした天の火だよ。ラーマヤーナではインドラの矢とも伝えてるがね。

▼ 表示結果

言葉を慎みたまえ！ 君はラピュタ王の前にいるのだ！

これから王国の復活を祝って、諸君にラピュタの力を見せてやろうと思ってね。見せてあげよう、ラピュタの雷を！

旧約聖書にあるソドムとゴモラを滅ぼした天の火だよ。ラーマヤーナではインドラの矢とも伝えているがね。

改行

改行は、無視されるか、半角空白と同じ扱いになります。正確には、次のような仕様です*1。

- 改行直前の文字と直後の文字のどちらかが日本語なら、改行は無視される。
- どちらも日本語でないなら、半角空白と同じ扱いになる。

次の例を見てください。

- 最初の段落では改行の前後（つまり行の終わりと行の始まり）がどちらも日本語なので、改行は無視されます。
- 2 番目の段落では改行の前後が日本語またはアルファベットなので、やはり改行は無視されます。
- 3 番目の段落では改行の前後がどちらもアルファベットなので、改行は半角空白扱いになります。

▼ サンプル

あいうえお
かきくけこ
さしすせそ。

たちつてとABC
なにぬねの
DEFはひふへほ。

まみむめもGHI
JKLらりるれろMNO
PQRやゆよ。

▼ 表示結果

あいうえおかきくけこさしすせそ。
たちつてと ABC なにぬねの DEF はひふへほ。
まみむめも GHI JKL らりるれろ MNO PQR やゆよ。

*1 なおこれは日本語 L^AT_EX の仕様です。

強制改行

強制的に改行したい場合は、「@
{ }」を使います。

▼ サンプル

```
土に根をおろし、風と共に生きよう。@<br>{ }
種と共に冬を越え、鳥と共に春をうたおう。
```

▼ 表示結果

土に根をおろし、風と共に生きよう。
種と共に冬を越え、鳥と共に春をうたおう。

なおこの例だと、「//noindent」をつけて段落の字下げをしないほうがいいでしょう。

▼ サンプル

```
//noindent
土に根をおろし、風と共に生きよう。@<br>{ }
種と共に冬を越え、鳥と共に春をうたおう。
```

▼ 表示結果

土に根をおろし、風と共に生きよう。
種と共に冬を越え、鳥と共に春をうたおう。

強制空行

強制的に空行を入れるには、「@
{ }」を連続してもいいですが、専用の命令「//blankline」を使うほうがいいでしょう。

▼ サンプル

```
//noindent
土に根をおろし、風と共に生きよう。

//blankline
//noindent
種と共に冬を越え、鳥と共に春をうたおう。
```

▼ 表示結果

土に根をおろし、風と共に生きよう。

種と共に冬を越え、鳥と共に春をうたおう。

改段落

インライン命令「@<par>{ }」を使うと、好きな箇所で改段落できます。

改段落は、通常は空行で行います。しかしたとえば箇条書きの中で改段落したい場合、安易に空行を入れるとそこで箇条書きが終了してしまいます。つまり「空行なら改段落する」という仕様は、箇条書きの文法とは相性が悪いのです。

このような場合は、「@<par>{ }」を使えば箇条書きの中でも改段落できます。

▼ サンプル

* 『貴官は自己の才能を示すのに、弁舌ではなく実績をもってすべきだろう。他人に命令するようなことが自分にもできるかどうか、やってみたらどうだ。』
@<par>{ }
富田耕生さん、今までありがとうございました。

▼ 表示結果

- 『貴官は自己の才能を示すのに、弁舌ではなく実績をもってすべきだろう。他人に命令するようなことが自分にもできるかどうか、やってみたらどうだ。』
富田耕生さん、今までありがとうございました。

なお箇条書きの中で改段落しても、インデントはされません*2。インデントしたい場合は、「@<par>{ i }」としてください。

▼ サンプル

* 『貴官は自己の才能を示すのに、弁舌ではなく実績をもってすべきだろう。他人に命令するようなことが自分にもできるかどうか、やってみたらどうだ。』
@<par>{ i }
富田耕生さん、今までありがとうございました。

▼ 表示結果

- 『貴官は自己の才能を示すのに、弁舌ではなく実績をもってすべきだろう。他人に命令するようなことが自分にもできるかどうか、やってみたらどうだ。』
富田耕生さん、今までありがとうございました。

改ページ

「//clearpage」で強制的に改ページできます。

なお Re:VIEW では「//pagebreak」で強制的に改ページできますが、これは L^AT_EX の

*2 これは L^AT_EX の仕様です。

「\pagebreak」命令を使うため、図や表のことが考慮されません。Starter では「//clearpage」を使ってください。

3.3 見出し

見出しレベル

章 (Chapter) や節 (Section) といった見出しは、「= 」や「== 」で始めます。

▼ サンプル

```
= 章(Chapter)見出し
== 節(Section)見出し
=== 項(Subsection)見出し
==== 目(Subsubsection)見出し
===== 段(Paragraph)見出し
===== 小段(Subparagraph)見出し
```

このうち、章・節・項・目はこの順番で（入れ子にして）使うことが想定されています。たとえば、章 (Chapter) の次に節ではなく項 (Subsection) が登場するのはよくありません。

ただし段と小段にはこのような制約はなく、たとえば目 (Subsubsection) を飛ばして節や項の中で使って構いません。

▼ サンプル

```
= 章(Chapter)
== 節(Section)
===== 段(Paragraph) ←節の中に段が登場しても構わない
```

Starter では、章は 1 ファイルにつき 1 つだけにしてください。1 つのファイルに複数の章を入れないでください。

▼ サンプル

```
= 章1
== 節
= 章2 ←（エラーにならないけど）これはダメ
```

見出し用オプション

見出しには次のようなオプションが指定できます。

▼ サンプル

```
==[nonum] 章番号や節番号をつけない（目次には表示する）
==[nodisp] 見出しを表示しない（目次には表示する）
==[notoc] 章番号や節番号をつけず、目次にも表示しない
```

一般的には、「まえがき」や「あとがき」には章番号をつけません。そのため、これらのオプションは「まえがき」や「あとがき」で使うとよさそうに見えます。

しかしこのようなオプションをつけなくても、「まえがき」や「あとがき」の章や節には番号がつきません。そのため、これらのオプションを使う機会は、実はほとんどありません。

章の概要と著者名

章タイトルの直後に、章の概要を書くことをお勧めします。

▼ サンプル

```
= チュートリアル

//abstract{
この章では、インストールから簡単な使い方までを説明します。
詳しい機能の説明は次の章で行います。
//}
```

章の概要は、たとえば次のように表示されます。本文と比べて小さめのゴシック体で、左右に余白が追加され、下に 2 行分空きます。

▼ 表示結果

この章では、インストールから簡単な使い方までを説明します。詳しい機能の説明は次の章で行います。

また、章の著者名を表記できます。複数の著者が集まって合同誌を書く場合に使うといいでしょう。著者名は章タイトルの直後、章の概要より前に書いてください。

▼ サンプル

```
//chapterauthor[ムスカ大佐]
```

▼ 表示結果

[著] ムスカ大佐

章 ID

章 (Chapter) のファイル名から拡張子の「.re」を取り除いた文字列を、「章 ID」と呼んでいます。たとえば、ファイル名が「02-tutorial.re」なら章 ID は「02-tutorial」、ファイル名が「preface.re」なら章 ID は「preface」です。

章 ID は、章そのものを参照する場合だけでなく、ある章から別の章の節 (Section) や図やテーブルを参照するときに使います。詳しくは後述します。

.....

章にラベルはつけられない

このあとで説明しますが、たとえば「=={ラベル タイトル}」のように書くと節 (Section) や項 (Subsection) にラベルがつけられます。

しかし「={ラベル タイトル}」のように書いても章にはラベルがつけられません。かわりに章 ID を使ってください。

.....

章を参照する

章 (Chapter) を参照するには、3 つの方法があります。

- 「@<chapref>{章 ID}」で章番号と章タイトルを参照できます (例: 「第 2 章 チュートリアル」)。
- 「@<chap>{章 ID}」で章番号を参照できます (例: 「第 2 章」)。
- 「@<title>{章 ID}」で章タイトルを参照できます (例: 「チュートリアル」)。

▼ サンプル

```
@<chapref>{02-tutorial}    ← 章番号と章タイトル
@<chap>{02-tutorial}      ← 章番号だけ
@<title>{02-tutorial}     ← 章タイトルだけ
```

▼ 表示結果

第 2 章 「チュートリアル」 ← 章番号と章タイトル
第 2 章 ← 章番号だけ
チュートリアル ← 章タイトルだけ

節や項を参照する

節 (Section) や項 (Subsection) を参照するには、まず節や項にラベルをつけます。

▼ サンプル

```
=={sec-heading} 見出し
==={subsec-headingref} 節や項を参照する
```

そして「@<hd>{ラベル}」を使ってそれらを参照します。

▼ サンプル

```
@<hd>{sec-heading}
@<hd>{subsec-headingref}
```

▼ 表示結果

「3.3 見出し」
「節や項を参照する」

また Starter 拡張である「@<secref>{ラベル}」を使うと、ページ番号も表示されます。コマンド名は「@<secref>」ですが、節 (Section) だけでなく項 (Subsection) や目 (Subsubsection) にも使えます。

▼ サンプル

```
@<secref>{sec-heading}
@<secref>{subsec-headingref}
```

▼ 表示結果

「3.3 見出し」 (p.46)
「節や項を参照する」 (p.49)

他の章の節や項（つまり他の原稿ファイルの節や項）を参照するには、ラベルの前に章 ID をつけます。

▼ サンプル

```
@<secref>{03-syntax|sec-heading}
@<secref>{03-syntax|subsec-headingref}
```

▼ 表示結果

「3.3 見出し」(p.46)

「節や項を参照する」(p.49)

なおラベルのかわりに節タイトルや項タイトルが使えますが、タイトルを変更したときにそれらを参照している箇所も書き換えなければならなくなるので、お勧めしません。一見面倒でも、参照先の節や項にラベルをつけることを強くお勧めします。

.....

項を参照するなら項番号をつけよう

デフォルトでは、章 (Chapter) と節 (Section) には番号がつきますが、項 (Subsection) には番号がつきません。そのため、「@<hd>{ }」で項を参照するとたとえば『「見出し用オプション」』となってしまい、項番号がないのでとても探しにくくなります。「@<secref>{ }」を使うと『3.1「見出し」の「見出し用オプション」』となるので多少ましですが、探しやすいとは言えません。

そのため、項 (Subsection) を参照したいなら項にも番号をつけましょう。config.yml の設定項目「secnolevel:」を「3」にすると、項にも番号がつくようになります。

.....

まえがき、あとがき、付録

「まえがき」や「あとがき」や「付録」の章は、catalog.yml で指定します*3。

▼ catalog.yml

```
PREDEF:
- 00-preface.re      ←まえがき

CHAPS:
- 01-install.re
- 02-tutorial.re
- 03-syntax.re

APPENDIX:
- 92-filelist.re     ←付録

POSTDEF:
- 99-postface.re     ←あとがき
```

「-」のあとに半角空白が必要なことに注意してください。またインデントにタブ文字は使わないでください。

*3 「catalog.yml」の中身は「YAML」という形式で書かれています。「YAML」を知らない人は Google 検索して調べてみてください。

部

「第 I 部」「第 II 部」のような部 (Part) を指定するには、`catalog.yml` で次のように指定します。

▼ catalog.yml

```
PREDEF:

CHAPS:
- 初級編:
  - chap1.re
  - chap2.re
- 中級編:
  - chap3.re
  - chap4.re
- 上級編:
  - chap5.re
  - chap6.re

APPENDIX:

POSTDEF:
```

この例では「第 I 部 初級編」「第 II 部 中級編」「第 III 部 上級編」の 3 つに分かれています。

部タイトルのあとには「:」をつけて「初級編:」のようにしてください。これを忘れると意味不明なエラーが出ます。

段見出しと小段見出し

Starter では、段 (Paragraph) 見出しは次のように表示されます。

▼ サンプル

```
テキストテキストテキストテキストテキストテキストテキストテキスト
テキストテキストテキストテキストテキストテキストテキストテキスト。
```

===== 段見出し1

```
テキストテキストテキストテキストテキストテキストテキストテキスト
テキストテキストテキストテキストテキストテキストテキストテキスト。
```

```
テキストテキストテキストテキストテキストテキストテキストテキスト
テキストテキストテキストテキストテキストテキストテキストテキスト。
```

===== 段見出し2

```
テキストテキストテキストテキストテキストテキストテキストテキスト
テキストテキストテキストテキストテキストテキストテキストテキスト。
```

テキストテキストテキストテキストテキストテキストテキストテキスト
テキストテキストテキストテキストテキストテキストテキストテキスト。

▼ 表示結果

テキストテキストテキストテキストテキストテキストテキストテキストテキストテキストテキスト
テキストテキストテキストテキストテキスト。

段見出し 1

テキストテキストテキストテキストテキストテキストテキストテキストテキストテキストテキスト
テキストテキストテキストテキストテキスト。

テキストテキストテキストテキストテキストテキストテキストテキストテキストテキストテキスト
テキストテキストテキストテキストテキスト。

段見出し 2

テキストテキストテキストテキストテキストテキストテキストテキストテキストテキストテキスト
テキストテキストテキストテキストテキスト。

テキストテキストテキストテキストテキストテキストテキストテキストテキストテキストテキスト
テキストテキストテキストテキストテキスト。

これを見ると分かるように、段見出しの前に少しスペースが入ります。しかし終わりにはそのようなスペースが入りません。終わりにスペースを入れるには、次のように「`//paragraphend`」を使ってください。

▼ サンプル

テキストテキストテキストテキストテキストテキストテキストテキスト
テキストテキストテキストテキストテキストテキストテキストテキスト。

===== 段見出し

テキストテキストテキストテキストテキストテキストテキストテキスト
テキストテキストテキストテキストテキストテキストテキストテキスト。
`//paragraphend`

テキストテキストテキストテキストテキストテキストテキストテキスト
テキストテキストテキストテキストテキストテキストテキストテキスト。

▼ 表示結果

テキストテキストテキストテキストテキストテキストテキストテキストテキストテキストテキスト
テキストテキストテキストテキストテキスト。

段見出し

テキストテキストテキストテキストテキストテキストテキストテキストテキストテキストテキスト

トテキストテキストテキストテキスト。

テキストテキストテキストテキストテキストテキストテキストテキストテキストテキストテキスト
トテキストテキストテキストテキストテキスト。

また、小段 (Subparagraph) 見出しは次のように表示されます。

▼ サンプル

テキストテキストテキストテキストテキストテキストテキストテキスト
テキストテキストテキストテキストテキストテキストテキストテキスト。

===== 小段見出し1

テキストテキストテキストテキストテキストテキストテキストテキスト
テキストテキストテキストテキストテキストテキストテキストテキスト。

テキストテキストテキストテキストテキストテキストテキストテキスト
テキストテキストテキストテキストテキストテキストテキストテキスト。

===== 小段見出し2

テキストテキストテキストテキストテキストテキストテキストテキスト
テキストテキストテキストテキストテキストテキストテキストテキスト。

テキストテキストテキストテキストテキストテキストテキストテキスト
テキストテキストテキストテキストテキストテキストテキストテキスト。

▼ 表示結果

テキストテキストテキストテキストテキストテキストテキストテキストテキストテキストテキスト
トテキストテキストテキストテキストテキスト。

小段見出し 1 テキストテキストテキストテキストテキストテキストテキストテキスト
テキストテキストテキストテキストテキストテキストテキスト。

テキストテキストテキストテキストテキストテキストテキストテキストテキストテキストテキスト
トテキストテキストテキストテキストテキスト。

小段見出し 2 テキストテキストテキストテキストテキストテキストテキストテキスト
テキストテキストテキストテキストテキストテキストテキスト。

テキストテキストテキストテキストテキストテキストテキストテキストテキストテキストテキスト
トテキストテキストテキストテキストテキスト。

これを見ると分かるように、小段見出しではスペースは入りません。「//subparagraph」は用意されていますが、スペースは入りません。

3.4 箇条書きと定義リスト

番号なし箇条書き

番号なし箇条書きは「 * 」で始めます。先頭に半角空白が入っていることに注意してください。

▼ サンプル

- * 箇条書き1
- * 箇条書き2
- * 箇条書き3

▼ 表示結果

- 箇条書き 1
- 箇条書き 2
- 箇条書き 3

「*」を連続させると、箇条書きを入れ子にできます。

▼ サンプル

- * 箇条書き
- ** 入れ子の箇条書き
- *** 入れ子の入れ子

▼ 表示結果

- 箇条書き
 - 入れ子の箇条書き
 - * 入れ子の入れ子

番号つき箇条書き

番号つき箇条書きは2つの書き方があります。

1つ目は「 1. 」のように始める書き方です。この方法は数字しか指定できず、また入れ子にできません。

▼ サンプル

1. 番号つき箇条書き（数字しか指定できない）
2. 番号つき箇条書き（入れ子にできない）

▼ 表示結果

1. 番号つき簡条書き（数字しか指定できない）
2. 番号つき簡条書き（入れ子にできない）

2つ目は Starter による拡張で、「 - 1. 」のように始める書き方です。この書き方は「 - A. 」や「 - (1) 」など任意の文字列が使え、また「-」を連続することで入れ子にもできます。

▼ サンプル

- (A) 番号つき簡条書き（英数字が使える）
- (B) 番号つき簡条書き（任意の文字列が使える）
- (B-1) 番号つき簡条書き（入れ子にできる）
- *** 番号なし簡条書きを含めることもできる

▼ 表示結果

- (A) 番号つき簡条書き（英数字が使える）
- (B) 番号つき簡条書き（任意の文字列が使える）
 - (B-1) 番号つき簡条書き（入れ子にできる）
 - 番号なし簡条書きを含めることもできる

どちらの記法も、先頭に半角空白が必要なことに注意してください。

.....

数字を使うべきか、文字を使うべきか

番号つき簡条書きでは、順番に強い意味がある場合は「1.」のように数字を使い、順番にさほど意味はなく単にラベルとして使いたい場合は「A.」のように文字を使います。詳しくは「簡条書きを正しく使い分ける」(p.187)を参照してください。

.....

定義リスト

HTML でいうところの「<dl><dt><dd>」は、「定義リスト」と「説明リスト」の2つがあります。前者は Re:VIEW でも使え、後者は Starter による拡張です。この節では前者の「定義リスト」を説明し、次の節で後者の「説明リスト」を説明します。なお会話形式の文章を作成するには、この節ではなく「3.16 会話形式」(p.114)を参照してください。

定義リストは、「：」で始めて、次の行からインデントします。

- 説明文のインデントは、半角空白でもタブ文字でもいいです。しかし混在させるとトラブルの元なので、どちらかに統一しましょう。
- 先頭の半角空白は入れるようにしましょう。過去との互換性のため、先頭の半角空白がなくても動作しますが、簡条書きとの整合性のために半角空白を入れましょう。

▼ サンプル

```
: 用語1
  説明文。
  説明文。
: 用語2
  説明文。
  説明文。
```

▼ 表示結果

用語 1

説明文。説明文。

用語 2

説明文。説明文。

説明文の中には、箇条書きが入れられます（Starter 拡張）。ただし、注意点が2つあります。

- 説明文の1行目を箇条書きで始めることはできません^{*4}。1行目は必ず通常の文章にしてください。
- 箇条書きを使うときは、インデントにはタブ文字を使わず半角空白を使ってください。半角空白4文字でインデントするといいいでしょう。

▼ サンプル

```
: 用語
  説明文。
  * 項目1
  * 項目1
```

▼ 表示結果

用語

説明文。

- 項目 1
 - 項目 1
-

説明文には、プログラムコードなどは入れられません。入れたい場合は、次で紹介する「説明リスト」を使います。

^{*4} この制約は、Re:VIEW の文法に起因します。

説明リスト

「説明リスト」(Description List) とは、先ほど説明した「定義リスト」(Definition List) の機能を拡張したものです。定義リストと比べて、説明リストには次のような特徴があります。

- キーや用語を太字にする/しないが選べる。
- キーや用語の直後で改行する/しないが選べる。
- 説明文の中に、簡条書きだけでなくプログラムコードなどを入れられる。
- 説明文の 1 行目から簡条書きを書ける。
- 専用の構文を使わないので、「：」だけで書ける定義リストと比べて記述が煩雑。
- 文章における役割は定義リストと同じ。

説明リストは次のように書きます。途中にある「@
{ }」は改行を表します。

▼ サンプル

```
//desclist{
//desc[開催日時]{
20XX年XX月XX日 10:00～12:00
//}
//desc[開催場所]{
XXXXXXXXXビルXXXXXXXXX会議室@<br>{ }
( XXXXXX市XXXXXX町XX-XX-XX )
//}
//}
```

▼ 表示結果

開催日時

20XX 年 XX 月 XX 日 10:00～12:00

開催場所

XXXXXXXXX ビル XXXXXXXXXX 会議室
(XXXXXXXX 市 XXXXXXXX 町 XX-XX-XX)

「：」だけで済む定義リストと比べて説明リストの書き方は煩雑ですが、表 3.1 のようなオプションを指定できるという利点があります。これらのオプションのデフォルト値は、「config-starter.yml」で設定されています。

オプションは、「//desclist」の第 1 引数に指定します。先ほどの例をオプションつきで表示してみましょう。ここで「7zw」は全角 7 文字分という意味です。

▼ サンプル

```
//desclist[bold=on,compact=on,indent=7zw]{
//desc[開催日時]{
```

▼ 表 3.1: 「//desclist」のオプション

オプション	説明	デフォルト値
bold=on	キーや用語を太字にする	off
compact=on	キーや用語の直後で改行しない	off
indent=幅	説明文のインデント幅	3zw
listmargin=高さ	説明リスト上下の空きの高さ	0.5zw
itemmargin=高さ	説明項目間の空きの高さ	0mm
classname=名前	HTML タグに追加するクラス名	

```
20XX年XX月XX日 10:00～12:00
//}
//desc[開催場所]{
XXXXXXXXXビルXXXXXXXXX会議室@<br>{}
( XXXXXX市XXXXXXXX町XX-XX-XX )
//}
//}
```

▼ 表示結果

開催日時 20XX 年 XX 月 XX 日 10:00～12:00
開催場所 XXXXXXXXXXX ビル XXXXXXXXXXX 会議室
 (XXXXXXXX 市 XXXXXXXX 町 XX-XX-XX)

説明リストでは、説明文の1行目から箇条書きが書けます。ただし次のサンプルを見れば分かるように、1行目が箇条書きだとスペースが空きます。

▼ サンプル

```
//desclist{
//desc[参加申込者]{
* アリス
* ボブ
* チャーリー
//}
//}
```

▼ 表示結果

参加申込者

- アリス
- ボブ
- チャーリー

この問題については将来対応する予定です。当面は、「`//vspace`」を使ってスペースを削除してください。

▼ サンプル

```
//desclist{
//desc[参加申込者]{
//vspace[latex][-5mm]
* アリス
* ボブ
* チャーリー
//}
//}
```

▼ 表示結果

参加申込者

- アリス
- ボブ
- チャーリー

また説明文の中に、プログラムコードなども入れられます。

▼ サンプル

```
//desclist{
//desc[print文]{
「@<code>|print()|」を使うと、文字列を表示できます。
//list{
def hello(name):
    print(f"Hello, {name}!")
//}
//}
//}
```

▼ 表示結果

print 文

「`print()`」を使うと、文字列を表示できます。

```
def hello(name):
    print(f"Hello, {name}!")
```


3.5 インライン命令とブロック命令

ここで少し話を変えて、命令の形式について説明します。

Re:VIEW および Starter でのコマンドは、大きく 3 つの形式に分けられます。

- インライン命令（例：「@{...}」）
- ブロック命令（例：「//list{...//}」）
- 特殊な形式の命令（例：箇条書きの「*」や見出しの「=」など）

ここではインライン命令とブロック命令について、それぞれ説明します。

インライン命令

「インライン命令」とは、「@{...}」のような形式の命令のことです。主に、文の途中に埋め込んで使います。

Re:VIEW のインライン命令は入れ子にできませんが、Starter では入れ子にできるよう拡張しています。

▼ サンプル

```
@<code>{func(@<b>{@<i>{arg}})}
```

▼ 表示結果

```
func(arg)
```

インライン命令は、複数行を対象にできません。たとえば次のような書き方はできません。

▼ サンプル

```
@<B>{テキスト
テキスト}。
```

インライン命令の中に「}」を書きたい場合は、次のうちどちらかを使います。

- 「@{var x = {a: 1\};}」のように「\}」とエスケープする。
- 「@{...}」のかわりに「@|...|」または「@\$...\$」を使う。

2 番目の書き方は、Re:VIEW では「フェンス記法」と呼ばれています。この記法では、バックスラッシュによるエスケープができないので注意してください。たとえば「@|\||」や「@\$\\$\$」のようなエスケープをしても効果はありません。

..... インライン命令の入れ子対応と複数の引数

インライン命令には、「@<href>{url, text}」や「@<yomi>{text, yomi}」のように複数の引数を受け取るものがあります。

しかしこの書き方は、実は入れ子のインライン命令と相性がよくありません。望ましいのは「@<href url="url">{text}」や「@<ruby yomi="yomi">{text}」のような形式であり、採用を現在検討中です。

.....

ブロック命令

「ブロック命令」とは、「//list{...//}」のような形式の命令のことです。

ブロック命令は、必ず行の先頭から始まる必要があります。たとえば「//list{」や「//}」の前に空白があると、ブロック命令として認識されません。

ブロック命令は入れ子にできます (Starter による拡張)。たとえば次の例では、「//note{...//}」の中に「//list{...//}」が入っています。

▼ サンプル

```
//note[ノートサンプル]{
//list[][フィボナッチ]{
def fib(n):
    return n if n <= 1 else fib(n-1)+fib(n-2)
//}
//}
```

ただし次のブロック命令はその性質上、中に他のブロック命令を入れられません (インライン命令は入れられます)。

- プログラムコードを表す「//list」
- ターミナルを表す「//terminal」と「//cmd」
- L^AT_EX や HTML の生テキストを埋め込む「//embed」

ブロック命令の引数は、たとえば「//list[ラベル][説明文]{...//}」のように書きます。この場合だと、第1引数が「ラベル」で第2引数が「説明文」です。引数の中に「】」を入れたい場合は、「\】」のようにエスケープしてください。

3.6 強調と装飾

強調

文章の一部を強調するには「@{...}」または「@{...}」で囲みます。囲んだ部分は太字のゴシック体になります。

▼ サンプル

```
テキスト@<B>{テキスト}テキスト。
```

▼ 表示結果

テキスト**テキスト**テキスト。

日本語の文章では、強調箇所は太字にするだけでなくゴシック体にするのが一般的です。そのため、強調したい場合は「@{...}」または「@{...}」を使ってください。

インライン命令

「@{...}」のような記法はインライン命令と呼ばれています。インライン命令は必ず 1 行に記述します。複数行を含めることはできません。

▼ このような書き方はできない

```
@<B>{テキスト
テキスト
テキスト。}
```

太字

太字にするだけなら「@{...}」で囲みます。強調と違い、ゴシック体になりません。

▼ サンプル

```
テキスト@<b>{テキスト}テキスト。
```

▼ 表示結果

テキスト**テキスト**テキスト。

前述したように、日本語の文章では強調するときは太字のゴシック体にするのが一般的です。そのため強調したいときは、ゴシック体にならない「@{...}」は使わないほうがいいでしょう。

ただし、プログラムコードの中では「@{...}」ではなく「@{...}」を使ってください。理由は、プログラムコードは等幅フォントで表示しているのに、「@{...}」だとゴシック体の

フォントに変更してしまうからです。「@{...}」はフォントを変更しないので、等幅フォントのまま太字になります。

装飾

テキストを装飾するには、表 3.2 のようなインライン命令を使います。

▼ 表 3.2: 装飾用のインライン命令

意味	入力	出力
太字	@{テキスト}	テキスト
強調	@{テキスト}	テキスト
強調	@{テキスト}	テキスト
傍点	@<bou>{テキスト}	テキスト
網掛け	@<ami>{テキスト}	テキスト
下線	@<u>{テキスト}	テキスト
取り消し線	@{テキスト}	テキスト
目立たせない	@<weak>{テキスト}	テキスト
ゴシック体	@{テキスト}	テキスト
イタリック体	@<i>{Text}	Text
等幅	@<tt>{Text}	Text
コード	@<code>{Text}	Text

文字サイズ

文字の大きさを変更するインライン命令もあります（表 3.3）。

▼ 表 3.3: 文字の大きさを変更するインライン命令

意味	入力	出力
小さく	@<small>{テキスト}	テキスト
もっと小さく	@<xsmall>{テキスト}	テキスト
もっともっと小さく	@<xxsmall>{テキスト}	テキスト
大きく	@<large>{テキスト}	テキスト
もっと大きく	@<xlarge>{テキスト}	テキスト
もっともっと大きく	@<xxlarge>{テキスト}	テキスト

マーキング用

装飾ではなく、論理的な意味を与えるマーキング用のインライン命令もあります（表 3.4）。追加されたテキストを表す「@<ins>{ }」は、通常の本文では下線になりますが、プログラムコード（//list{ ... //}）やターミナル（//terminal{ ... //}）の中では太字になります。

▼ 表 3.4: マーキング用のインライン命令

意味	入力	出力
追加されたテキスト	@<ins>{xxx}	<u>xxx</u>
プログラムコード	@<code>{xxx}	xxx
ファイル名	@<file>{xxx}	xxx
ユーザ入力文字列	@<userinput>{xxx}	<u>xxx</u>

.....

「@<tt>{ }」ではなく「@<code>{ }」や「@<file>{ }」を使う

プログラムコードやコマンド文字列を等幅フォントで表すには、「@<tt>{ }」ではなく「@<code>{ }」を使ってください。理由は、プログラムコードであることを表すにはフォントを変更するコマンドではなく、「プログラムコードである」ことを表すコマンドを使うべきだからです。加えて両者ではシングルクォートとバッククォートの表示が異なり、たとえば「@<tt>{ ' }」は「'」と表示され、「@<code>{ ' }」は「'」と表示されます（Starter 拡張）^{*5}。

同じような理由で、ファイル名を表すには「@<file>{ }」を使ってください。「@<tt>{ }」や「@<code>{ }」を使うべきではありません。

.....

その他のインライン命令

ダブルクォーテーション

- 「@<qq>{ ... }」を使うと、引数の文字列を適切なダブルクォーテーションで囲みます。
- PDF では「 `` ... ' ' 」のように囲みます。これは \LaTeX での標準的な方法であり、表示がきれいです。
 - HTML では「 “ ... ” 」のように全角記号で囲みます。
- 表示を比べてみましょう。
- 「@<qq>{Apple}」は、「“Apple”」と表示されます。
 - 「"Apple"」（半角記号）は、「"Apple"」と表示されます。
 - 「 “Apple”」（全角記号）は、「“Apple”」と表示されます。

^{*5} 使用する等幅フォントによっては同じように表示されます。

3.7 プログラムリスト

基本的な書き方

プログラムリストは「`//list{ ... //}`」で囲み、第2引数に説明書きを指定します。

▼ サンプル

```
//list[][フィボナッチ数列]{
def fib(n):
    return n if n <= 1 else fib(n-1) + fib(n-2)
//}
```

▼ 表示結果

▼ フィボナッチ数列

```
def fib(n):
    return n if n <= 1 else fib(n-1) + fib(n-2)
```

もし説明文字列の中に「`]`」を含める場合は、「`\]`」のようにエスケープしてください。また説明書きをつけない場合は、引数を省略して「`//list{ ... //}`」のように書けます。

リスト番号

第1引数にラベル名を指定すると、リスト番号がつきます。また「`@<list>{ラベル名}`」とすると、ラベル名を使ってプログラムリストを参照できます。そのため、ラベル名は他と重複しない文字列にしてください。

▼ サンプル

サンプルコードはこちら（`@<list>{fib2}`）。

```
//list[fib2][フィボナッチ数列]{
def fib(n):
    return n if n <= 1 else fib(n-1) + fib(n-2)
//}
```

▼ 表示結果

サンプルコードはこちら（リスト 3.1）。

▼ リスト 3.1: フィボナッチ数列

```
def fib(n):
    return n if n <= 1 else fib(n-1) + fib(n-2)
```

第 1 引数だけ指定して「`//list[ラベル名]{ ... //}`」のようにすると、リスト番号はつくけど説明はつきません。

ラベルの自動指定

リスト番号はつけたいけど、重複しないラベル名をいちいちつけるのが面倒なら、ラベル名かわりに「?」を指定します。ただし「`@<list>{ラベル名}`」での参照はできなくなります。

▼ サンプル

```
//list[?]{
def fib(n):
    return n if n <= 1 else fib(n-1) + fib(n-2)
//}
```

▼ 表示結果

▼ リスト 3.2:

```
def fib(n):
    return n if n <= 1 else fib(n-1) + fib(n-2)
```

この機能は、すべてのプログラムリストに手っ取り早くリスト番号をつけたいときに便利です。

長い行の折り返し

Starter では、プログラムコードの長い行は自動的に折り返されます。行が折り返されると行末と次の行の先頭に折り返し記号がつくので、どの行が折り返されか簡単に見分けがつきます。

▼ サンプル

```
//list{
sys.stderr.write("Something error raised. Please contact to system admini>
>strator.")
//}
```

▼ 表示結果

```
sys.stderr.write("Something error raised. Please contact to system admini>
>strator.")
```

「`@{}`」で太字にしたり、「`@{}`」で取り消し線を引いても、きちんと折り返しされます。

▼ サンプル

```
//list{
@<b>{sys.stderr.write("Something error raised. Please contact to system a
> dministrator.")}
@<del>{sys.stderr.write("Something error raised. Please contact to system
> administrator.")}
//}
```

▼ 表示結果

```
sys.stderr.write("Something error raised. Please contact to system admini
> strator.")
sys.stderr.write("Something error raised. Please contact to system admini
> strator.")
```

ただし折り返し箇所が日本語だと、折り返しされても折り返し記号がつきません。折り返し箇所が日本語でも、折り返し記号がつくようになりました。以前のバージョンを使っている場合は Starter のプロジェクトを作り直してみてください。

折り返しをしたくないときは、「`//list[]][fold=off]`」と指定します。

▼ サンプル

```
//list[]][fold=off]{
sys.stderr.write("Something error raised. Please contact to system admini
> strator.")
//}
```

▼ 表示結果

```
sys.stderr.write("Something error raised. Please contact to system admini
> strator.")
```

折り返しはするけど折り返し記号をつけたくない場合は、「`//list[]][foldmark=off]`」と指定します。

▼ サンプル

```
//list[]][foldmark=off]{
sys.stderr.write("Something error raised. Please contact to system admini
> strator.")
//}
```

▼ 表示結果

```
sys.stderr.write("Something error raised. Please contact to system administrator.")
```

改行文字を表示

第3引数に「[eolmark=on]」と指定をすると、改行文字がうっすらと表示されます。この機能は折り返し記号をオフにしてから使うといいでしょう。

▼ サンプル

```
//list[][][eolmark=on,foldmark=off]{
function fib(n) {
    if (n <= 1) { return n; }
    return fib(n-1) + fib(n-2);
}
//}
```

▼ 表示結果

```
function fib(n) {
    if (n <= 1) { return n; }
    return fib(n-1) + fib(n-2);
}
```

.....
折り返し記号や改行文字をマウス選択の対象外にする

PDF 中のプログラムコードをマウスでコピーしたとき、折り返し記号や改行文字がコピーした文字列に含まれると、それらをいちいち除去しないといけないので不便です。

Starter では、これらの記号をマウス選択の対象から外すことができます。詳しくは「[PDF] 折り返し記号や行番号をマウス選択の対象から外す」(p.149) を参照してください。

.....

インデント

第3引数にたとえば「[indent=4]」*6のような指定をすると、4文字幅でのインデントを表す縦線がうっすらとつきます。

*6 後方互換性のために、「[indentwidth=4]」という名前でも指定できます。

▼ サンプル

```
//list[][][indent=4]{
class Fib:

    def __call__(n):
        if n <= 1:
            return n
        else:
            return fib(n-1) + fib(n-2)

//}
```

▼ 表示結果

```
class Fib:

    def __call__(n):
        if n <= 1:
            return n
        else:
            return fib(n-1) + fib(n-2)
```

Python のようにインデントの深さでブロックを表すプログラミング言語では、ブロックの終わりが明示されません。そのためプログラムの途中で改ページされると、その前後のブロックの関係が読者には分からなくなります。

そのような場合は、この機能を使ってインデントを表示してあげるとよいでしょう。

.....

インデント記号をコピー対象にしない

以前の Starter では、インデントを表す記号としてパイプ記号「|」を使っていました。そのせいで、プログラムコードをマウスでコピーするとパイプ記号も一緒にコピーされてしまっていました。これは読者にとってうれしくない仕様です。

現在の Starter では、インデントを表す記号のかわりに図形として縦線を引いています。そのおかげで、コピーしてもパイプ記号が入りません^{*7}。Starter による気配りの一例です。

.....

外部ファイルの読み込み

プログラムコードを外部ファイルから読み込むには、2 つ方法があります。

^{*7} ただし、Starter のデフォルトでは折り返し記号や行番号がコピー対象に含まれてしまいます。これについては「[PDF] 折り返し記号や行番号をマウス選択の対象から外す」(p.149) を参照してください。

ひとつは、「`//list[][][file=...]`」のように「`//list`」の第3引数にファイル名を指定することです。この場合、ブロックの中身は無視されます（ブロックそのものは省略できません）。

▼ サンプル

```
//list[][][file=source/fib3.rb]{
//}
```

もうひとつは、「`@<include>{}`」というインライン命令を使うことです*⁸。

▼ サンプル

```
//list[fib3][フィボナッチ数列]{
@<include>{source/fib3.rb}
//}
```

脚注のリンク先にあるように、この方法は Re:VIEW では undocumented です。Starter では正式な機能としてサポートします。

タブ文字

プログラムコードの中にタブ文字があると、8文字幅のカラムに合うよう自動的に半角空白に展開されます。

しかし「`@{...}`」のようなインライン命令があると、カラム幅の計算が狂うため、意図したようには表示されないことがあります。

そのため、プログラムコードにはなるべくタブ文字を含めないほうがいいでしょう。

全角文字の幅を半角 2 文字分にする

プログラムコードのデフォルトでは、全角文字の幅は半角文字 2 文字分より少し狭くなっています。「`//list`」の第3引数に「`widecharfit=on`」というオプションを指定すると、全角文字の幅が半角文字 2 文字分に揃います。

▼ サンプル

```
//list[][][widecharfit=on]{
123456789_123456789_123456789_123456789_123456789_
あいうえおかきくけこさしすせそたちつてとなにぬねの
//}
```

▼ 表示結果

*⁸ 参考: <https://github.com/kmuto/review/issues/887>

```
123456789_123456789_123456789_123456789_123456789_
あいうえおかきくけこさしすせそたちつてとなにぬねの
```

プログラムやターミナル画面において、全角と半角が混在しているせいで表示が崩れる場合は、このオプションを指定すると表示が崩れなくなります。

▼ 「wcharfit=on」がない場合：表示が崩れる

```
testdb1=> select * from members order by id;
 id |  name  | height | gender
-----+-----+-----+-----
 101 | エレン |    170 | M
 102 | ミカサ |    170 | F
 103 | アルミン |    163 | M
 104 | ジャン |    175 | M
 105 | サシャ |    168 | F
 106 | コニー |    158 | M
(6 rows)
```

▼ 「wcharfit=on」がある場合：表示が崩れない

```
testdb1=> select * from members order by id;
 id |  name  | height | gender
-----+-----+-----+-----
 101 | エレン |    170 | M
 102 | ミカサ |    170 | F
 103 | アルミン |    163 | M
 104 | ジャン |    175 | M
 105 | サシャ |    168 | F
 106 | コニー |    158 | M
(6 rows)
```

このオプションをデフォルトで有効にしたい場合は、「オプションのデフォルト値」(p.148)を参考にしてください。

.....

全角文字の判定

全角文字の判定は、現在は次のようなコードで行っており、かなりいい加減です。改善については相談してください。

▼ 全角文字の判定方法 (Ruby)

```
string.each_char do |char| # charは長さ1の文字列
  if char =~ /\[\000-\177]/
    # 半角文字
```



```

else
  # 全角文字
end
end
end

```

.....

出力結果

出力結果や変換結果を表すための「`//output`」というブロック命令も用意されています。使い方は「`//list`」と同じで、見た目が少し違うだけです。たとえば自動生成された HTML や CSS を表示したり、SQL とその実行結果を分けて表示したいときに使うといいでしょう。

▼ サンプル

```

//list[][一覧を取得するSQL]{
select gender, count(*)
from members
group by gender
order by gender;
//}

//output[][実行結果]{
gender | count
-----+-----
F      |      2
M      |      4
(2 rows)
//}

```

▼ 表示結果

▼ 一覧を取得する SQL

```

select gender, count(*)
from members
group by gender
order by gender;

```

▼ 実行結果

```

gender | count
-----+-----
F      |      2
M      |      4
(2 rows)

```

「`//output`」では全角文字の幅が半角文字のちょうど2個分になるよう設定されているので、半角と全角が混在していても表示が崩れません*9。

▼ 「`//list`」での表示結果（少し崩れる）

```
id | name | height | gender
-----+-----+-----+-----
101 | エレン | 170 | M
102 | ミカサ | 170 | F
103 | アルミン | 163 | M
(3 rows)
```

▼ 「`//output`」での表示結果（崩れない）

```
id | name | height | gender
-----+-----+-----+-----
101 | エレン | 170 | M
102 | ミカサ | 170 | F
103 | アルミン | 163 | M
(3 rows)
```

その他のオプション

「`//list`」の第3引数には、次のようなオプションも指定できます。

`fontsize={small|x-small|xx-small|large|x-large|xx-large}`

文字の大きさを小さく（または大きく）します。どうしてもプログラムコードを折返ししたくないときに使うといいでしょう。

`lineno={on|off|integer}`

行番号をつけます。詳細は次の節で説明します。

`linenowidth={0|integer}`

行番号の幅を指定します。詳細は次の節で説明します。

`classname=classname`

HTML タグのクラス名を指定します。HTML または ePub のときだけ機能します。

`lang=langname`

プログラム言語の名前を指定します。コードハイライトのために使いますが、Starter ではまだ対応していません。

なおオプションのデフォルト値は、「`config-starter.yml`」で設定できます。詳しくは「オプションのデフォルト値」(p.148) を参照してください。

*9 ただし、「`//output`」で使われる等幅フォントによっては表示が崩れることがあります。その場合は `config-starter.yml` で「`output_ttfont: beramono`」を指定してみてください。

3.8 行番号

プログラムリストに行番号をつける

プログラムリストに行番号をつけるには、「`//list`」ブロック命令の第3引数を次のように指定します。

▼ サンプル

```
//list[][][lineno=on]{  
function fib(n) {  
    return n <= 1 ? n : fib(n-1) + fib(n-2);  
}  
//}
```

▼ 表示結果

```
1 function fib(n) {  
2     return n <= 1 ? n : fib(n-1) + fib(n-2);  
3 }
```

「on」のかわりに開始行番号を指定できます。

▼ サンプル

```
//list[][][lineno=99]{  
function fib(n) {  
    return n <= 1 ? n : fib(n-1) + fib(n-2);  
}  
//}
```

▼ 表示結果

```
99 function fib(n) {  
100     return n <= 1 ? n : fib(n-1) + fib(n-2);  
101 }
```

「`lineno=1`」のかわりに「1」とだけ書いても、行番号がつきます。行番号をつける方法としてはいちばん短い書き方です。

▼ サンプル

```
//list[][][1]{  
function fib(n) {  
    return n <= 1 ? n : fib(n-1) + fib(n-2);  
}  
//}
```

▼ 表示結果

```
1 function fib(n) {  
2     return n <= 1 ? n : fib(n-1) + fib(n-2);  
3 }
```

行番号の幅を指定する

前の例では行番号が外側に表示されました。行番号の幅を指定すると、行番号が内側に表示されます。

▼ サンプル

```
//list[][][lineno=on,linewidth=3]{  
function fib(n) {  
    return n <= 1 ? n : fib(n-1) + fib(n-2);  
}  
//}
```

▼ 表示結果

```
1: function fib(n) {  
2:     return n <= 1 ? n : fib(n-1) + fib(n-2);  
3: }
```

また幅を「0」に指定すると、行番号の幅を自動的に計算します。

▼ サンプル

```
//list[][][lineno=99,linewidth=0]{  
function fib(n) {  
    return n <= 1 ? n : fib(n-1) + fib(n-2);  
}  
//}
```

▼表示結果

```

99: function fib(n) {
100:   return n <= 1 ? n : fib(n-1) + fib(n-2);
101: }

```

複雑な行番号指定

行番号は「`lineno=80-83&97-100&105-`」のような複雑な指定もできます。

- 「80-83」は80行目から83行目を表します。
- 「105-」は105行目以降を表します。
- 「&」は行番号をつけないことを表します。

▼サンプル

```

//list[][][lineno=80-83&97-100&105-,linenowidth=0]{
// JavaScript
function fib(n) {
  return n <= 1 ? n : fib(n-1) + fib(n-2);
}
... (省略) ...
## Ruby
def fib(n)
  n <= 1 ? n : fib(n-1) + fib(n-2);
end
... (省略) ...
## Python
def fib(n):
  return n if n <=1 else fib(n-1) + fib(n-2)
//}

```

▼表示結果

```

80: // JavaScript
81: function fib(n) {
82:   return n <= 1 ? n : fib(n-1) + fib(n-2);
83: }
   : ... (省略) ...
97: ## Ruby
98: def fib(n)
99:   n <= 1 ? n : fib(n-1) + fib(n-2);
100: end

```

```
    : ... (省略) ...  
105: ## Python  
106: def fib(n):  
107:     return n if n <=1 else fib(n-1) + fib(n-2)
```

3.9 ターミナル

基本的な書き方

ターミナル（端末）の画面を表すには、「`//terminal{ ... //}`」というブロック命令を使います。

▼ サンプル

```
//terminal[?][PDFを生成]{
$ rake pdf          @<balloon>{Dockerを使わない場合}
$ rake docker:pdf    @<balloon>{Dockerを使っている場合}
//}
```

▼ 表示結果

▼ リスト 3.3: PDF を生成

```
$ rake pdf          ← Dockerを使わない場合
$ rake docker:pdf    ← Dockerを使っている場合
```

引数はプログラムリスト「`//list{ ... //}`」と同じなので、引数の説明は省略します。

過去との互換性のために、「`//cmd{ ... //}`」というブロック命令もあります。しかしこの命令はリスト番号や行番号がつけられないし、「`//list`」命令と使い方が違うので、もはや使う必要はありません。

▼ サンプル

```
//cmd[PDFを生成]{
$ rake pdf          @<balloon>{Dockerを使わない場合}
$ rake docker:pdf    @<balloon>{Dockerを使っている場合}
//}
```

▼ 表示結果

▼ PDF を生成

```
$ rake pdf          ← Dockerを使わない場合
$ rake docker:pdf    ← Dockerを使っている場合
```

ユーザ入力

「@<userinput>{ }」を使うと、ユーザによる入力箇所を示せます*10。

▼ サンプル

```
//terminal[?][PDFを生成]{
$ @<userinput>{rake pdf}           @<balloon>{Dockerを使わない場合}
$ @<userinput>{rake docker:pdf}    @<balloon>{Dockerを使っている場合}
//}
```

▼ 表示結果

▼ リスト 3.4: PDF を生成

```
$ rake pdf           ← Dockerを使わない場合
$ rake docker:pdf    ← Dockerを使っている場合
```

ターミナルのカーソル

「@<cursor>{...}」を使うと、ターミナルでのカーソルを表せます。

次の例では、2行目の真ん中の「f」にカーソルがあることを表しています。

▼ サンプル

```
//terminal{
function fib(n) {
  return n <= 1 ? n : @<cursor>{f}ib(n-1) : fib(n-2);
}
~
~
"fib.js" 3L, 74C written
//}
```

▼ 表示結果

```
function fib(n) {
  return n <= 1 ? n : fib(n-1) : fib(n-2);
}
~
~
"fib.js" 3L, 74C written
```

*10 ただし、「@<userinput>{ }」では長い行を自動的に折り返せません。これは今後の課題です。

3.10 脚注

脚注の書き方

脚注は、脚注をつけたい箇所に「@<fn>{ラベル}」を埋め込み、脚注の文は「//footnote[ラベル][脚注文]」のように書きます。

▼ サンプル

本文。本文@<fn>{fn-101}。本文。

//footnote[fn-101][脚注文。脚注文。]

▼ 表示結果

本文。本文^{*11}。本文。

このページの最下部に脚注が表示されていることを確認してください。

また脚注文に「】」を埋め込む場合は、「\】」のようにエスケープしてください。

▼ サンプル

本文@<fn>{fn-102}。

//footnote[fn-102][脚注に「\】」を入れる。]

▼ 表示結果

本文^{*12}。

脚注の注意点

L^AT_EX の制限により、脚注が表示されなかったりエラーになることがあります。

- コラムの中で脚注を使うと、脚注が表示されないことがあります。この場合、コラムを明示的に閉じてください。詳しくは「コラム内の脚注」(p.89)を参照してください。
- ミニブロックの中で脚注を使うと、脚注が表示されないことがあります。ミニブロックについての詳細は「ノート以外のミニブロック」(p.85)を参照してください。

*11 脚注文。脚注文。

*12 脚注に「】」を入れる。

.....

脚注がうまく表示されない理由

コラムやミニブロックでは、囲み枠が使われています。L^AT_EX で囲み枠を実現するには、「minipage 環境」という機能を使うのが一般的です。L^AT_EX の minipage 環境はページ内で別のページを実現する機能であり、HTML でいうなら iframe のようなものです。

そして minipage 環境はそれ自体が独立したページなので、minipage 環境内で脚注を使うとそれが外側のページにうまく伝わりません。これがコラムやミニブロックで脚注がうまく使えない理由です。

対策としては、コラムやミニブロックのデザイを枠線を使わないように変更することです。または、コラムやミニブロックでは脚注を使わないように文章を工夫することです。

なお Starter では、ミニブロックのうちノート (//note) に関しては minipage 環境を使わないデザインにしています。ノートについては次の節で紹介します。

.....

3.11 ノート

補足情報や注意事項などを表示する枠を、Starter では「ノート」と呼びます。

ノートの書き方

ノートは、「`//note{ ... //}`」というブロック命令で表します。

▼ サンプル

```
//note[締め切りを守るたったひとつの冴えたやり方]{
原稿の締め切りを守るための、素晴らしい方法を教えましょう。
それは、早い時期に執筆を開始することです。
//}
```

▼ 表示結果

```
.....
締め切りを守るたったひとつの冴えたやり方
    原稿の締め切りを守るための、素晴らしい方法を教えましょう。それは、早い時期に執筆を
    開始することです。
.....
```

ノートには、箇条書きやプログラムコードを入れられます（Starter 独自拡張）。

▼ サンプル

```
//note[ノートタイトル]{
* 箇条書き
* 箇条書き

//list[][プログラムコード]{
def fib(n):
    return n if n <= 1 else fib(n-1) + fib(n-2)
//}
//}
```

▼ 表示結果

```
.....
ノートタイトル
    • 箇条書き
    • 箇条書き
.....
```

▼ プログラムコード

```
def fib(n):
    return n if n <= 1 else fib(n-1) + fib(n-2)
```

.....

なお現在、ノートのタイトルに「@<code>{...}」を入れると「TeX capacity exceeded, sorry [input stack size=5000].」という L^AT_EX のエラーが発生します。原因は不明なので、もしこのようなエラーが発生したらノートのタイトルから「@<code>{...}」を外してみてください。

ラベルをつけて参照する

ノートにラベルをつけて、「@<noteref>{ラベル名}」で参照できます (Starter 独自拡張)。

▼ サンプル

```
//note[note-123][詳細情報]{
  詳細な説明は次のようになります。……
//}
```

詳しくは@<noteref>{**note-123**}を参照してください。

▼ 表示結果

.....

詳細情報

詳細な説明は次のようになります。……

.....

詳しくはノート「詳細情報」(p.84)を参照してください。

上の例で、第1引数がノートタイトルではなくラベル名になっていることに注意してください。通常は次のどちらかを使うといいでしょう。

- タイトルだけを指定するなら「//note[タイトル]{...//}」
- ラベルとタイトルを指定するなら「//note[ラベル][タイトル]{...//}」

.....

ノートブロック命令における引数の詳しい仕様

「//note」ブロック命令は、過去との互換性を保ったままラベル機能を導入したので、引数の仕様が少し複雑になっています。

▼ 表 3.5: 「//note」の引数の仕様

記述	ラベル	タイトル
//note[ラベル][タイトル]	あり	あり
//note[][タイトル]	なし	あり
//note[ラベル][]	あり	なし
//note[タイトル]	なし	あり
//note[][]	なし	なし
//note[]	なし	なし
//note	なし	なし

引数の詳しい仕様を表 3.5 にまとめました。これを見ると分かるように、**第 1 引数だけを指定した場合はラベルではなくタイトルとみなされます**。注意してください。

.....

ノート以外のミニブロック

ノート以外にも、Re:VIEW では次のようなブロックが用意されています。

- 「//memo」(メモ)
- 「//tip」(Tips)
- 「//info」(情報)
- 「//warning」(警告)
- 「//important」(重要事項)
- 「//caution」(注意喚起)
- 「//notice」(お知らせ)

これらを Re:VIEW で使うとひどいデザインで表示されますが、Starter では簡素なデザインで表示するよう修正しています。

▼ サンプル

```
//notice[ご来場のみなさまへ]{
本建物の内部には、王族しか入ることができません。
それ以外の方は入れないので、あらかじめご了承ください。
//}
```

▼ 表示結果

ご来場のみなさまへ

本建物の内部には、王族しか入ることができません。それ以外の方は入れないので、あらかじめご了承ください。

第1引数のタイトルは省略できます。

また Starter では、「`//info`」と「`//caution`」と「`//warning`」に専用のアイコン画像^{*13}を用意しています。

▼ サンプル

```
//info[解決のヒント]{
```

本製品を手に取り、古くから伝わるおまじないを唱えてみましょう。

すると天空の城への門が開きます。

```
//}
```

```
//caution[使用上の注意]{
```

本製品を石版にかざすと、天空の城から雷が発射されます。

周りに人がいないことを確かめてから使用してください。

```
//}
```

```
//warning[重大な警告]{
```

本製品を持ったまま滅びの呪文を唱えると、天空の城は自動的に崩壊します。

大変危険ですので、決して唱えないでください。

```
//}
```

▼ 表示結果



解決のヒント

本製品を手に取り、古くから伝わるおまじないを唱えてみましょう。すると天空の城への門が開きます。



使用上の注意

本製品を石版にかざすと、天空の城から雷が発射されます。周りに人がいないことを確かめてから使用してください。



重大な警告

本製品を持ったまま滅びの呪文を唱えると、天空の城は自動的に崩壊します。大変危険ですので、決して唱えないでください。

^{*13} これらのアイコン画像はパブリックドメインにするので、商用・非商用を問わず自由にお使いください。再配布も自由に行って構いません。

これらのアイコン画像は Keynote でささっと作ったものなので、もっといいデザインのアイコン画像に変更するといいいでしょう。デザインの変更方法は「[PDF]」「//info」や「//warning」のデザインを変更する」(p.143)を参照してください。

なお「//note」と違い、いくつか制限事項があります。

- 参照用のラベルはつけられません。
- L^AT_EX の制限により、脚注がうまく表示されません。
- L^AT_EX の制限により、プログラムリストやターミナル画面がページをまたげません。
- L^AT_EX の制限により、画像と表の位置指定（「[pos=h]」のような指定）ができません。また強制的に「pos=H」が指定されたものとして扱われます。

.....

「warning」と「caution」の違い

「warning」と「caution」は、どちらも警告や注意を表す言葉ですが、次のような違いがあるそうです^{*14}。

- ・「warning」は致命的な結果になる可能性がある場合に使う。
- ・「caution」は望ましくない結果になる可能性がある場合に使う。

つまり「warning」は強い警告、「caution」は弱い警告だと思えばいいでしょう。

.....

^{*14} 参考：『Technical Writing: The Difference Between Warnings and Cautions』（<http://www.stevensstrategic.com/technical-writing-the-difference-between-warnings-and-cautions/>）

3.12 コラム

コラムとは

コラムは、補足情報や関連情報を記述する囲みです。ノートとよく似ていますが、次のような違いがあります。

- ノートは主に数行～十数行程度の内容です。コラムはもっと長い内容に使うことが多いです。
- ノートは目次に出ません。コラムは目次に出ます^{*15}。
- ノートは文章のどこに書いても構いません。コラムは主に章 (Chapter) の最後に書くことが多いです。

ときどき、ノートで書くべき内容をすべてコラムとして書いている同人誌を見かけます。間違いだとは言いきれないのですが、できれば両者の使い分けを意識しましょう。

コラムの書き方

コラムは「`==[column]`」と「`==[/column]`」か、または「`===[column]`」と「`===[/column]`」で囲みます (イコール記号の数が違うことに注意)。また実装上の都合により、コラムを閉じる直前に空行を入れてください。

▼ サンプル

===[column] サンプルコラム1

コラム本文。

- a. 箇条書き
- b. 箇条書き

```
//list[][プログラムリスト]{
```

```
def fib(n):
```

```
    return n if n <= 1 else fib(n-1) + fib(n-2)
```

```
//}
```

←空行を入れてから、

===[/column]

←コラムを閉じる

▼ 表示結果

【コラム】 サンプルコラム 1

^{*15} ただし「`[notoc]`」オプションをつけた場合を除く。

コラム本文。

- a. 箇条書き
- b. 箇条書き

▼ プログラムリスト

```
def fib(n):
    return n if n <= 1 else fib(n-1) + fib(n-2)
```

コラム内の脚注

前の例を見れば分かるように、コラムの中には箇条書きやプログラムリストなどを含めることができます。ただし、脚注だけはコラムを閉じたあとに書く必要があります。

▼ サンプル

```
===[column] サンプルコラム2
コラム本文@<fn>{fn-222}。

===[/column]                                ←コラムを閉じて、

//footnote[fn-222][脚注の文章。] ←そのあとに書くこと！
```

▼ 表示結果

【コラム】 サンプルコラム 2

コラム本文^{*16}。

コラムを参照

コラムを参照するには、コラムにラベルをつけて、それを「@<column>{ラベル}」で参照します。

^{*16} 脚注の文章。

▼ サンプル

```
===[column]{column3} サンプルコラム3
サンプルコラム本文。

===[/column]                                ←コラムを閉じて、

@<column>{column3}を参照。
```

▼ 表示結果

【コラム】 サンプルコラム 3

サンプルコラム本文。

コラム 「サンプルコラム 3」を参照。

コラムの制限

現在、コラムには次のような制限があります。

- コラム内で「//list」や「//terminal」を使うと、それらはページをまたいでくれません^{*17}。
- 画像と表の位置指定ができません。「//image[][][pos=h]」や「//table[][][pos=h]」を指定するとエラーになるので^{*18}、指定しないでください。

これらの制限は仕様です。あきらめてください。

.....

コラムは見出しの一種

Re:VIEW および Starter では、コラムは節 (Section) や項 (Subsection) といった見出しの一種として実装されています。そのため、「==[column]」なら節 (Section) と同じような扱いになり、「===[column]」なら項 (Subsection) と同じような扱いになります (目次レベルを見るとよく分かります)。

また節や項は閉じる必要がない (勝手に閉じてくれる) のと同じように、コラムも「==[/column]」がなければ勝手に閉じてくれます。しかし明示的に閉じないと脚注が出力されないことがあるので、横着をせずに明示的にコラムを閉じましょう。

.....

^{*17} これは L^AT_EX の framed 環境による制限です。解決するには framed 環境を使わないよう変更する必要があります。
^{*18} oframe 環境の中では table 環境や figure 環境のようなフローティング要素が使えないという L^AT_EX の制約が原因です。

3.13 画像

画像ファイルの読み込み方

画像を読み込むには、「`//image[画像ファイル名][説明文字列][オプション]`」を使います。

- 画像ファイルは「`images/`」フォルダに置きます^{*19}。
- 画像ファイル名には拡張子を指定しません（自動的に検出されます）。
- 画像ファイル名がラベルを兼ねており、「`@{画像ファイル名}`」で参照できます。
- 画像の表示幅は第3引数のオプションとして指定します。詳しくは後述します。
- 第3引数のオプションは省略できます。どんなオプションが指定できるかは後述します。

次の例では、画像ファイル「`images/tw-icon1.jpg`」を読み込んでいます。

▼ サンプル

```
//image[tw-icon1][Twitterアイコン][width=30mm]
```

表示例は図 3.1 をご覧ください。



▲ 図 3.1: Twitter アイコン

1 つの画像ファイルを複数の箇所から読み込まない

「`//image`」コマンドでは、画像ファイル名がラベルを兼ねます。そのせいで、1 つの画像ファイルを複数の箇所から読み込むとラベルが重複してしまい、コンパイル時に警告が出ます。また「`@{ }`」で参照しても、画像番号がずれてしまい正しく参照できません。

Re:VIEW および Starter では 1 つの画像ファイルを複数の箇所から読み込むのは止めておきましょう。かわりにファイルをコピーしてファイル名を変えましょう^{*20}。

画像の表示幅を指定する

「`//image`」の第3引数に、画像を表示する幅を指定できます。指定方法は2つあります。

^{*19} 画像ファイルを置くフォルダは、設定ファイル「`config.yml`」の設定項目「`imagedir`」で変更できますが、通常は変更する必要はないでしょう。

^{*20} （分かる人向け）もちろんシンボリックリンクやハードリンクでもいいです。

```
//image[][][scale=0.5]
```

画像の幅が本文幅より大きい小さいかで挙動が変わります。

- 画像の幅が本文幅より大きい場合は、本文幅を基準とします。この例なら、本文幅の半分の幅で表示されます。
- 画像の幅が本文幅より小さい場合は、画像の幅を基準とします。この例なら、画像幅の半分の幅で表示されます。

```
//image[][][width=50%]
```

常に本文幅の半分の幅で画像を表示します。また「50%」のような比率だけでなく、「150px」や「50mm」のように本文幅と関係ない長さの指定ができます。値には必ず単位が必要であり、「width=0.5」のような指定はできません。

「scale」も「width」も指定しなかった場合は、次のように表示されます。

- 画像の幅が本文幅より大きい場合は、本文幅に縮小されて表示されます。
- 画像の幅が本文幅より小さい場合は、画像幅のまま表示されます^{*21}。

なお「scale=0.5」は Re:VIEW でも Starter でも使えるオプションですが、「width=50%」は Starter でのみ使えるオプションです。「width=50%」は Re:VIEW では使えないので注意してください。

PDF と HTML の場合で表示幅を変えるような指定もできます。「latex::width=100%, html::width=50%」^{*22}のように指定すると、PDF のときは本文幅いっぱいに画像を表示し、HTML のときは本文幅の半分の幅で画像を表示します。

.....

「scale=0.5」の挙動について

Re:VIEW のドキュメント^{*23}には、「scale=0.5」について次のように説明されています。

3 番目の引数として、画像の倍率・大きさを指定できます。今のところ「scale=X」で倍率 (X 倍) を指定でき、HTML、TeX ともに紙面 (画面) 幅に対しての倍率となります (0.5 なら半分の幅になります)。

しかし、これは画像の幅が本文幅より大きい場合の説明です。画像の幅が本文より小さい場合は、「scale=0.5」は本文幅の半分ではなく画像幅の半分になります。つまり、こういうことです：

- 画像の幅が本文幅より大きい場合は、本文幅を基準とする。
- 画像の幅が本文幅より小さい場合は、画像幅を基準とする。

ややこしいことに、Re:VIEW が HTML を生成するときは必ず本文幅を基準としてしまいます。なぜなら、Re:VIEW では画像の表示幅を指定するのに「width:50%」のような CSS を使うからです。

^{*21} Re:VIEW で HTML を生成すると、画像の幅が本文幅より小さい場合、本文幅いっぱいに表示されてしまいます。この挙動は困るので、Starter では画像幅のまま表示するよう修正しています。

^{*22} この指定方法は、他のオプションや他のブロックコマンドでも使用できます。

^{*23} <https://github.com/kmuto/review/blob/master/doc/format.ja.md#図>

これは PDF のときと挙動が大きく違うので、Starter ではかわりに「max-width:50%」のような CSS を使い、PDF の挙動に似せています。ただし完全に同じ動作にはならないので、どうしても Re:VIEW と互換性を持たせる必要がある場合を除き、「scale=0.5」ではなく「width=50%」を使うことを勧めます。

.....

章ごとの画像フォルダ

画像ファイルは、章 (Chapter) ごとのフォルダに置けます。たとえば原稿ファイル名が「02-tutorial.re」であれば、章 ID は「02-tutorial」なので、画像ファイルを「images/02-tutorial/」に置けます*24（特別な設定は不要です）。これは複数の著者で一冊の本を執筆するときに便利です。

詳しくは Re:VIEW のマニュアル*25を参照してください。

画像のまわりに線をつける

第 3 引数に「[border=on]」をつけると、画像のまわりに灰色の線がつきます (Starter 拡張)。

▼ サンプル

```
//image[tw-icon2][Twitterアイコン][scale=0.3,border=on]
```

表示結果は図 3.2 をご覧ください。



▲ 図 3.2: Twitter アイコン

画像の配置位置を指定する

第 3 引数に「[pos=...]」をつけると、読み込んだ画像をページのどこに配置するか、指定できます (Starter 拡張)。

pos=H

なるべく現在位置に配置します。現在位置に読み込めるスペースがなければ次のページの先頭に配置し、現在位置にはスペースが空いたままになります。Re:VIEW のデフォルトですが、ページ数が無駄に増えるのでお勧めしません。

*24 もちろん「images/」にも置けます。

*25 <https://github.com/kmuto/review/blob/master/doc/format.ja.md%E7%94%BB%E5%83%8F%E3%83%95%E3%82%A1%E3%82%A4%E3%83%AB%E3%81%AE%E6%8E%A2%E7%B4%A2>

pos=h

上と似ていますが、画像が次のページの先頭に配置されたときは、現在位置に後続の文章が入るため、スペースが空きません。Starter のデフォルトです*26。

pos=t

ページ先頭に配置します。

pos=b

ページ最下部に配置します。

pos=p

独立したページに画像だけを配置します（後続の文章が入りません）。大きな画像はこれで表示するといいでしょう。

▼ サンプル

現在位置に配置

```
//image[tw-icon][Twitterアイコン][scale=0.3,pos=h]
```

ページ先頭に配置

```
//image[tw-icon][Twitterアイコン][scale=0.3,pos=t]
```

ページ最下部に配置

```
//image[tw-icon][Twitterアイコン][scale=0.3,pos=b]
```

「pos=H」と「pos=h」の違いは、図 3.3 を見ればよく分かるでしょう。

ただし位置指定は、コラムの中では使えず、ノート以外のミニブロック（「//info」や「//warning」など）の中でも使えません。どちらの場合も、位置指定は強制的に「pos=H」と同じ動作になります。これは L^AT_EX の制約からくる仕様です。

画像に番号も説明もつけない

今まで説明したやり方では、画像に番号と説明文字列がつけました。これらをつけず、単に画像ファイルを読み込みたいだけの場合は、「//indepimage[ファイル名][][scale=1.0]」を使ってください。第2引数に説明文字列を指定できますが、通常は不要でしょう。

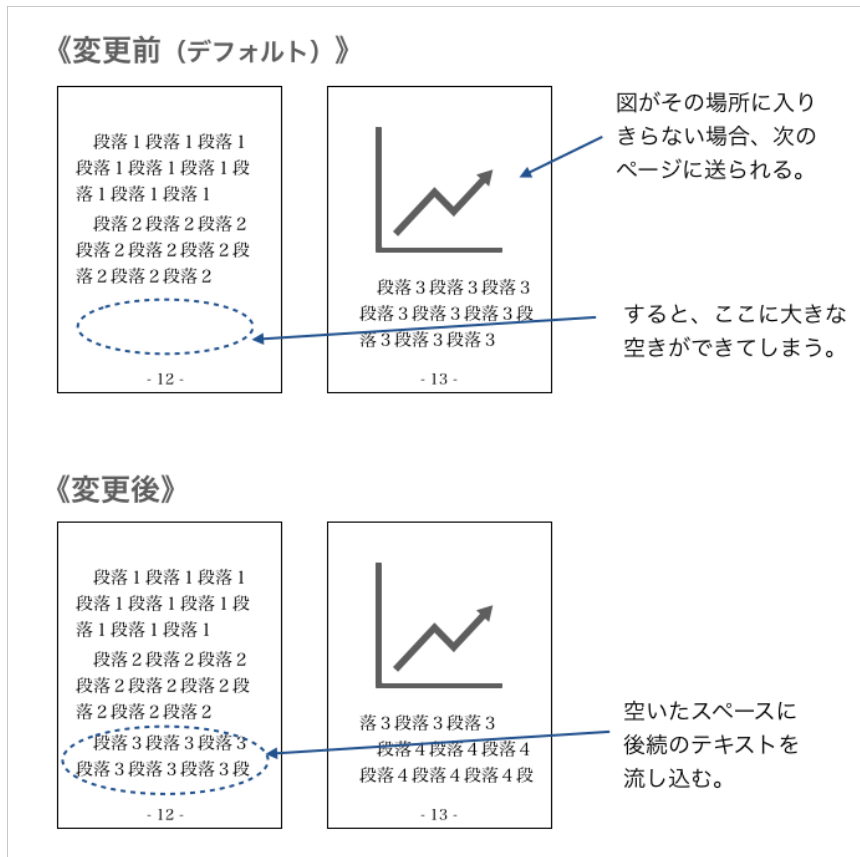
▼ サンプル

```
//indepimage[tw-icon3][][scale=0.3]
```

▼ 表示結果

第2引数に説明文字列を指定した場合、Re:VIEW では「図：」という接頭辞がつけます。しかし//indepimageの用途を考えると、この接頭辞は邪魔でしょう。そこで Starter では、このよう

*26 設定ファイル「config-starter.yml」の設定項目「image_position:」でデフォルト値を変更できます。



▲ 図 3.3: 「pos=H」(上) と「pos=h」(下) の違い



な接頭辞をつけないようにしました。もし「図:」が必要なら自分で説明文字列につけてください。

文章の途中に画像を読み込む

文章の途中で画像を読み込むには、「@<icon>{画像ファイル名}」を使います。画像の表示幅は指定できないようです。

▼ サンプル

文章の途中でファビコン画像「@<icon>{favicon-16x16}」を読み込む。

▼ 表示結果

文章の途中でファビコン画像「」を読み込む。

画像とテキストを並べて表示する

Starter では、画像とテキストを並べて表示するためのコマンド「`//sideimage`」を用意しました。著者紹介において Twitter アイコンとともに使うといいでしょう。

▼ サンプル

```
//sideimage[tw-icon4][20mm][side=L,sep=7mm,border=on]{
//noindent
@<B>{カウプラン機関極東支部}

* @_kauplan (@<href>{https://twitter.com/_kauplan/})
* @<href>{https://kauplan.org/}
* 技術書典8新刊「Pythonの黒魔術」出ました。

//}
```

▼ 表示結果



カウプラン機関極東支部

- @_kauplan (https://twitter.com/_kauplan/)
- <https://kauplan.org/>
- 技術書典 8 新刊「Python の黒魔術」出ました。

使い方は「`//sideimage[画像ファイル][画像表示幅][オプション]{ ... //}`」です。

- 画像ファイルは「`//image`」と同じように指定します。
- 画像表示幅は「`30mm`」「`3.0cm`」「`1zw`」「`10%`」などのように指定します。使用できる単位はこの4つであり、「`1zw`」は全角文字1文字分の幅、「`10%`」は本文幅の10%になります。なお「`//image`」と違い、単位がない「`0.1`」が10%を表すという仕様ではなく、エラーになります。
- オプションはたとえば「`side=L,sep=7mm,boxwidth=40mm,border=on`」のように指定します。
 - 「`side=L`」で画像が左側、「`side=R`」で画像が右側にきます。デフォルトは「`side=L`」。
 - 「`sep=7mm`」は、画像と本文の間のセパレータとなる余白幅です。デフォルトはなし。
 - 「`boxwidth=40mm`」は、画像を表示する領域の幅です。画像表示幅より広い長さを指定してください。デフォルトは画像表示幅と同じです。
 - 「`border=on`」は、画像を灰色の線で囲みます。デフォルトは `off`。

なお「`//sideimage`」は内部で $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ の `minipage` 環境を使っているため、次のような制限が

あります。

- 途中で改ページされません。
- 画像下へのテキストの回り込みはできません。
- 脚注が使えません。

こういった制限があることを分かったうえで使ってください。

3.14 表

表の書き方

表（テーブル）は「`//table[ラベル][説明文字列]{ ... //}`」のように書きます。

- ラベルを使って「`@table{ラベル}`」のように参照できます。
- セルは 1 文字以上のタブ文字で区切ります。
- ヘッダは 12 文字以上の「-」か「=」で区切ります。

▼ サンプル

```
@<table>{tbl-sample1}を参照。

//table[tbl-sample1][テーブルサンプル]{
Name      Val      Val      Val
-----
AAA       10       10       10
BBB       100      100      100
CCC       1000     1000     1000
//}
```

▼ 表示結果

表 3.6 を参照。

▼ 表 3.6: テーブルサンプル

Name	Val	Val	Val
AAA	10	10	10
BBB	100	100	100
CCC	1000	1000	1000

ラベルの自動採番

「`//table[][]`」の第 1 引数はラベルです。これがないと、「表 1.1」のような表示がされません。しかし、重複しない文字列を用意して第 1 引数に設定するのは、結構面倒くさいです。

そこで Starter では、ラベルのかわりに「?」を指定する機能を用意しました。これを使うと、ランダム文字列が自動的に割り当てられるため、いつも「表 1.1」のような表示がされます（これは「`//list[?]`」と同じ機能です）。

次の例では、「`//table`」の第 1 引数がラベルではなく「?」になっています。これで重複しないランダム文字列が自動的に割り当てられるので、この表は番号つきで表示されます。

▼ サンプル

```
//table[?][キャプション]{
Name      Val      Val      Val
-----
AAA       10       10       10
BBB       100      100      100
CCC       1000     1000     1000
//}
```

▼ 表示結果

▼ 表 3.7: キャプション

Name	Val	Val	Val
AAA	10	10	10
BBB	100	100	100
CCC	1000	1000	1000

右寄せ、左寄せ、中央揃え

セルの右寄せ (r)・左寄せ (l)・中央揃え (c) を指定するには、「//tsize[latex][...]」*27 を使います。ただし現在のところ、この機能は PDF 用であり ePub では使えません。

▼ サンプル

```
//tsize[latex][|l|r|l|c|]
//table[tbl-sample2][右寄せ(r)、左寄せ(l)、中央揃え(c)]{
Name      Val      Val      Val
-----
AAA       10       10       10
BBB       100      100      100
CCC       1000     1000     1000
//}
```

▼ 表示結果

「//tsize」の使い方は2種類あります。

- (A) 「//tsize[|latex||r|l|c|]」 … Re:VIEW での書き方 (Starter でも使える)
- (B) 「//tsize[latex][|r|l|c|]」 … Starter で追加した書き方 (Re:VIEW では使えない)

*27 「//tsize[pdf][...]」ではなく「//tsize[latex][...]」なのは、内部で L^AT_EX という組版用ソフトウェアを使っているからです。

▼ 表 3.8: 右寄せ (r)、左寄せ (l)、中央揃え (c)

Name	Val	Val	Val
AAA	10	10	10
BBB	100	100	100
CCC	1000	1000	1000

(A) の書き方では、「`latex`」や「`html`」を表す縦線と、罫線を指定する縦線とが混在しており、とても分かりにくいです。これに対し、Starter で追加された (B) の書き方であれば、ずっと見やすく書けます。どうしても Re:VIEW との互換性が必要な場合を除き、(B) の書き方をお勧めします。

なお「`latex`」を指定しない場合は次のように書きます。

- 「`//tsize[|||r|l|c|]`」… Re:VIEW での書き方
- 「`//tsize[]|l|r|l|c|]`」… Starter で追加した書き方
- 「`//tsize[*]|l|r|l|c|]`」… これでもよい

また「`//tsize[html,epub][...]`」のような指定もできます。将来的に「`//tsize`」が HTML や ePub をサポートすれば、このような指定が役に立つでしょう。

縦の罫線

「`//tsize[]|l|r|l|c|]`」の指定において、「`|`」は縦方向の罫線を表します。これをなくすと罫線がつきません。また「`||`」のように二重にすると、罫線も二重になります。

▼ サンプル

```
//tsize[]|l||rlc|
//table[tbl-sample3][罫線をなくしたり、二重にしてみたり]{
Name      Val      Val      Val
-----
AAA       10       10       10
BBB       100      100      100
CCC       1000     1000     1000
//}
```

▼ 表示結果

▼ 表 3.9: 罫線をなくしたり、二重にしてみたり

Name	Val	Val	Val
AAA	10	10	10
BBB	100	100	100
CCC	1000	1000	1000

横の罫線

Re:VIEW では、表のどの行にも横の罫線が引かれます。Starter では、横の罫線を引かないように指定できます。そのためには、第 3 引数に「**hline=off**」を指定します。

▼ サンプル

```
//tsize[][lrlc]
//table[tbl-sample4][横の罫線を引かない][hline=off]{
Name      Val      Val      Val
=====
AAA       10       10       10
BBB      100      100      100
CCC     1000     1000     1000
//}
```

▼ 表示結果

▼ 表 3.10: 横の罫線を引かない

Name	Val	Val	Val
AAA	10	10	10
BBB	100	100	100
CCC	1000	1000	1000

「**hline=off**」を指定したとき、空行があるとそこに罫線が引かれます。

▼ サンプル

```
//tsize[][lrlc]
//table[tbl-sample5][空行が横の罫線になる][hline=off]{
Name      Val      Val      Val
=====
AAA       10       10       10
BBB      100      100      100

CCC     1000     1000     1000
DDD     1000     1000     1000

EEE     1000     1000     1000
FFF     1000     1000     1000
//}
```

▼ 表示結果

▼ 表 3.11: 空行が横の罫線になる

Name	Val	Val	Val
AAA	10	10	10
BBB	100	100	100
CCC	1000	1000	1000
DDD	1000	1000	1000
EEE	1000	1000	1000
FFF	1000	1000	1000

必要な箇所だけに罫線を引くと、表が見やすくなります。次の例を参考にしてください。

▼ サンプル

```
//tsize[][llr]
//table[tbl-sample6][出版社別コミックス発行部数][hline=off]{
出版社  タイトル      発行部数
=====
集英社  鬼滅の刃      XXXX万部
.       ワンピース      XXX万部
.       ハイキュー!!   XXX万部
.       呪術廻戦        XXX万部

講談社  進撃の巨人      XXX万部
.       FAIRY TAIL     XXX万部

小学館  名探偵コナン     XXX万部
.       銀の匙          XXX万部
//}
```

表示結果は表 3.12 を見てください。

セル幅

セルの幅を指定するには、「//tsize[][...]」の中でたとえば「p{20mm}」のように指定します。

- この場合、自動的に左寄せになります。右寄せや中央揃えにはできません。
- セル内の長いテキストは自動的に折り返されます。表が横にはみ出してしまう場合はセル幅を指定するといいいでしょう。

▼ 表 3.12: 出版社別コミックス発行部数

出版社	タイトル	発行部数
集英社	鬼滅の刃	XXXX 万部
	ワンピース	XXX 万部
	ハイキュー!!	XXX 万部
	呪術廻戦	XXX 万部
講談社	進撃の巨人	XXX 万部
	FAIRY TAIL	XXX 万部
小学館	名探偵コナン	XXX 万部
	銀の匙	XXX 万部

▼ サンプル

```
//tsize[latex][|l|p{70mm}|]
//table[tbl-sample7][セルの幅を指定すると、長いテキストが折り返される]{
Name      Description
-----
AAA      text text text text text text text text text text text text >
>text text
BBB      text text text text text text text text text text text text >
>text text
//}
```

▼ 表示結果

▼ 表 3.13: セルの幅を指定すると、長いテキストが折り返される

Name	Description
AAA	text text text text text text text text text text text text text text text
BBB	text text text text text text text text text text text text text text text text

空欄

セルを空欄にするには、セルに「.」だけを書いてください。

▼ サンプル

```
//tsize[latex][|l|ccc|]
//table[tbl-sample8][セルを空欄にするサンプル]{
評価項目      製品A   製品B   製品C
-----
```

機能が充分か	✓	✓	.
価格が適切か	.	.	✓
サポートがあるか	✓	.	✓
//}			

表示結果は表 3.14 をご覧ください。

▼ 表 3.14: セルを空欄にするサンプル

評価項目	製品 A	製品 B	製品 C
機能が充分か	✓	✓	
価格が適切か			✓
サポートがあるか	✓		✓

またセルの先頭が「.」で始まる場合は、「..」のように 2 つ書いてください。そうしないと先頭の「.」が消えてしまいます。

▼ サンプル

```
//table[tbl-sample9][セルの先頭が「.<code>{.}」で始まる場合][headerrows=0>
]>]{
先頭に「.」が1つだけの場合      .bashrc
先頭に「.」が2つある場合        ..bashrc
//}
```

表示結果は表 3.15 をご覧ください。

▼ 表 3.15: セルの先頭が「.」で始まる場合

先頭に「.」が 1 つだけの場合	bashrc
先頭に「.」が 2 つある場合	.bashrc

表示位置の指定

表を表示する位置を、「//table」の第 3 引数で指定できます。

▼ サンプル

```
//table[tbl-sample10][表示位置を指定][pos=ht]{
....
//}
```

指定できる文字は次の通りです。また複数の文字を指定できます。

- pos=h here（現在の場所）
- pos=H here forcedly（現在の場所に強制）

pos=t top (ページ上部)
 pos=b bottom (ページ下部)
 pos=p page (その表だけのページ)

ただし位置指定は、コラムの中では使えず、ノート以外のミニブロック(「`//info`」や「`//warning`」など)の中でも使えません。どちらの場合も、強制的に「`pos=H`」と同じ動作になります。これは \LaTeX の制約からくる仕様です。

表のフォントサイズ

表のフォントサイズを、「`//table`」の第3引数で指定できます。表が大きくて1ページに収まらないとき使うといいでしょう^{*28}。

▼ サンプル

```
//table[tbl-sample11][フォントを小さく][fontsize=small]{
Name      Val      Val      Val
=====
AAA       10       10       10
BBB      100      100      100
//}

//table[tbl-sample12][フォントをもっと小さく][fontsize=x-small]{
Name      Val      Val      Val
=====
AAA       10       10       10
BBB      100      100      100
//}

//table[tbl-sample13][フォントをもっと小さく][fontsize=xx-small]{
Name      Val      Val      Val
=====
AAA       10       10       10
BBB      100      100      100
//}
```

▼ 表示結果

指定できるフォントサイズは次の通りです。

^{*28} PDF では表のフォントサイズがデフォルトで `small` になっています。そのため、「`//table[...][fontsize=small]`」を指定しても小さくなりません。かわりに「`x-small`」を指定してください。

▼ 表 3.16: フォントを小さく

Name	Val	Val	Val
AAA	10	10	10
BBB	100	100	100

▼ 表 3.17: フォントをもっと小さく

Name	Val	Val	Val
AAA	10	10	10
BBB	100	100	100

▼ 表 3.18: フォントをもーっと小さく

Name	Val	Val	Val
AAA	10	10	10
BBB	100	100	100

▼ 表 3.19: 表のフォントサイズ

フォントサイズ	説明
medium	本文と同じサイズ
small	少し小さく
x-small	小さく
xx-small	かなり小さく
large	少し大きく
x-large	大きく
xx-large	かなり大きく

CSV 形式の表

「//table」の第 3 引数に「csv=on」を指定すると、表を CSV 形式で書けます（Starter 拡張）。

▼ サンプル

```
//table[tbl-csv1][CSVテーブル][csv=on]{
Name,Val,Val,Val
=====
AAA,10,10,10
BBB,100,100,100
CCC,1000,1000,1000
//}
```

▼ 表示結果

タブ区切りのときと同じように、12 個以上の「=」でヘッダを区切ります。
CSV 形式では、空欄はそのまま空欄として表示されます。「.」を入力する必要はありません。

▼表 3.20: CSV テーブル

Name	Val	Val	Val
AAA	10	10	10
BBB	100	100	100
CCC	1000	1000	1000

また「"..."」のようにすると、1つのセルに複数行の文字列を書けます。ただし複数行で書いても表示は1行になります。

▼ サンプル

```
//table[tbl-csv2][CSVテーブル][csv=on]{
ぼくはか,"ぼくのとなりに
暗黒破壊神がいます"
はめふら,"乙女ゲームの破滅フラグしかない
悪役令嬢に転生してしまった…"
//}
```

▼ 表示結果

▼表 3.21: CSV テーブル

ぼくはか	ぼくのとなりに暗黒破壊神がいます
はめふら	乙女ゲームの破滅フラグしかない悪役令嬢に転生してしまった…

表示も複数行にしたい場合は、明示的に改行します。

▼ サンプル

```
//table[tbl-csv3][CSVテーブル][csv=on]{
ぼくはか,"ぼくのとなりに@<br>{
暗黒破壊神がいます"
はめふら,"乙女ゲームの破滅フラグしかない@<br>{
悪役令嬢に転生してしまった…"
//}
```

▼ 表示結果

外部ファイルを読み込む

「//table」の第3引数に「file=filename」を指定すると、外部ファイルを読み込んでそれを表（テーブル）にできます（Starter 拡張）。

▼ 表 3.22: CSV テーブル

ぼくはか	ぼくのとなりに 暗黒破壊神がいます
はめふら	乙女ゲームの破滅フラグしかない 悪役令嬢に転生してしまった…

- ファイル名の拡張子が「.csv」なら、CSV 形式で読み込まれます。
- 第 3 引数に「csv=on」が指定された場合も、CSV 形式で読み込まれます。
- それ以外の場合は、タブ区切り形式で読み込まれます。

次の例では、「data/data1.csv」というファイルを読み込んで表にしています。ブロック（「{」から「//}」まで）は省略できないので注意してください。

▼ サンプル

```
//table[tbl-csv4][サンプルデータ][file=data/data1.csv]{  
//}
```

文字コードを指定して読み込むこともできます。指定できる文字コードは、「utf-8」「sjis」「shift_jis」「cp932」などです。デフォルトは UTF-8 です。

▼ サンプル

```
//table[tbl-csv5][サンプルデータ][file=data/data1.csv, encoding=sjis]{  
//}
```

ヘッダの行数を指定する

今までは、ヘッダを指定するには 12 個以上の「=」や「-」からなる区切り行を挿入する必要があります。しかし一般的な CSV ファイルでは、このような区切り行は入れず、かわりに最初の行をヘッダ行と見なすことがほとんどではないでしょうか。

「//table」では第 3 引数に「headerrows=1」を指定すると、区切り行がなくても最初の行をヘッダとして扱ってくれます。

▼ サンプル

```
//table[tbl-csv6][先頭行をヘッダとみなす][csv=on, headerrows=1]{  
Name,Val,Val,Val  
AAA,10,10,10  
BBB,100,100,100  
CCC,1000,1000,1000  
//}
```

▼ 表示結果

▼ 表 3.23: 先頭行をヘッダとみなす

Name	Val	Val	Val
AAA	10	10	10
BBB	100	100	100
CCC	1000	1000	1000

また「headerrows=0」を指定すると、ヘッダ行のない表として表示されます。

▼ サンプル

```
//table[tbl-csv7][3×3の魔方陣][csv=on,headerrows=0]{
6,1,8
7,5,3
2,9,4
//}
```

▼ 表示結果

▼ 表 3.24: 3 × 3 の魔方陣

6	1	8
7	5	3
2	9	4

ヘッダのカラム数を指定する

「//table[][][headercols=2]」とすると、左から2つのカラムをヘッダとして表示します。

▼ サンプル

```
//table[tbl-csv8][左2つのカラムをヘッダとみなす][csv=on,headercols=2]{
Name,Val,Val,Val
AAA,10,10,10
BBB,100,100,100
CCC,1000,1000,1000
//}
```

▼ 表示結果

このオプションは、ヘッダの行数を指定する「headercols=1」と共存できます。

▼ 表 3.25: 左 2 つのカラムをヘッダとみなす

Name	Val	Val	Val
AAA	10	10	10
BBB	100	100	100
CCC	1000	1000	1000

▼ サンプル

```
//table[tbl-csv9][上と左をヘッダにする][csv=on,headerrows=1,headercols=1]{
Name,Val,Val,Val
AAA,10,10,10
BBB,100,100,100
CCC,1000,1000,1000
//}
```

▼ 表示結果

▼ 表 3.26: 上と左をヘッダにする

Name	Val	Val	Val
AAA	10	10	10
BBB	100	100	100
CCC	1000	1000	1000

表を画像で用意する

Re:VIEW や Starter では、あまり複雑な表は作成できません。複雑な表は画像として用意し、それを読み込むのがいいでしょう。

「//imgtable」コマンドを使うと、画像を表として読み込みます。使い方は「//image」と同じです。

▼ サンプル

```
//imgtable[order-detail][複雑な表の例：注文詳細][scale=0.5]{
//}
```

▼ 表示結果

▼ 表 3.27: 複雑な表の例：注文詳細

注文番号	#XXXX	注文日	XXXX年XX月XX日	
顧客名	アルフォンス・エルリック			
注文明細	商品名	数量	金額	
(1)	水	35 <i>L</i>	¥ XXX	
(2)	アンモニア	4 <i>L</i>	¥ XXX	
(3)	石灰	1.5 <i>kg</i>	¥ XXX	
(4)	リン	800 <i>g</i>	¥ XXX	
(5)	塩分	250 <i>g</i>	¥ XXX	
(6)	硝石	100 <i>g</i>	¥ XXX	
合計				¥ XXX

3.15 数式

数式の書き方

数式は「`//texequation[ラベル][説明文]{ ... //}`」のように書きます。

▼ サンプル

```
//texequation[euler1][オイラーの公式]{
e^{i\theta} = \sin{\theta} + i\cos{\theta}
//}
```

▼ 表示結果

式 3.1: オイラーの公式

$$e^{i\theta} = \sin \theta + i \cos \theta$$

数式の書き方が独特に見えますが、これは \LaTeX での書き方です。 \LaTeX における数式の書き方は、Wikibooks^{*29}など他の資料をあたってください。

数式は、ラベルを使って「`@<eq>{ラベル}`」のように参照できます。たとえば「`@<eq>{euler1}`」とすると「式 3.1」となります。

ラベルや説明文がいらなければ、「`//texequation{ ... //}`」のように省略できます。

数式を文中に埋め込む

数式を文中に埋め込むには「`@<m>$....$`」を使います。

▼ サンプル

```
オイラーの公式は@<m>$e^{i\theta} = \sin{\theta} + i\cos{\theta}$です。
特に、@<m>${\theta} = \pi$のときは@<m>$e^{i\pi} = -1$となり、これはオイラーの等式
と呼ばれます。
```

▼ 表示結果

オイラーの公式は $e^{i\theta} = \sin \theta + i \cos \theta$ です。特に、 $\theta = \pi$ のときは $e^{i\pi} = -1$ となり、これはオイラーの等式と呼ばれます。

数式のフォント

数式中では、アルファベットはイタリック体になります。これは数式の伝統です。次の例で、

^{*29} <https://en.wikibooks.org/wiki/LaTeX/Mathematics>

「@<m>\$...\$」を使った場合と使わない場合を比べてください。

▼ サンプル

* @<m>\$y = f(x) + k\$	← 使った場合
* y = f(x) + k	← 使わなかった場合

▼ 表示結果

- $y = f(x) + k$ ← 使った場合
- $y = f(x) + k$ ← 使わなかった場合

その他、Re:VIEW における数式の詳細は、Re:VIEW の Wiki^{*30}を参照してください。

^{*30} <https://github.com/kmuto/review/blob/master/doc/format.ja.md#tex-%E5%BC%8F>

3.16 会話形式

Starter では、会話形式の文章を書けます*³¹ (Starter 拡張)。

アイコン画像を使う場合

会話形式は次のようにして書きます。

- 全体を「`//talklist{ ... //}`」で囲う。
- 発言を「`//talk[avatar-b]{ ... //}`」で囲う。
- 「`//talk`」の第1引数にアイコン画像ファイルを指定する（「`//image`」の第1引数と同じ）。

▼ サンプル

```
//talklist{
//talk[avatar-b]{
できもしないことをおっしゃらないでください。
//}
//talk[avatar-g]{
不可能なことを言い立てるのは貴官の方だ。
それも安全な場所から動かずにな。
//}
//talk[avatar-b]{
……小官を侮辱なさるのですか。
//}
//talk[avatar-g]{
貴官は自己の才能を示すのに、弁舌ではなく実績をもってすべきだろう。
他人に命令するようなことが自分にもできるかどうか、やってみたらどうだ。
//}
//}
```

▼ 表示結果



「できもしないことをおっしゃらないでください。」



「不可能なことを言い立てるのは貴官の方だ。それも安全な場所から動かずにな。」

*³¹ LINE のようなチャット形式はサポートしていません。



「……小官を侮辱なさるのですか。」



「貴官は自己の才能を示すのに、弁舌ではなく実績をもってすべきだろう。他人に命令するようなことが自分にもできるかどうか、やってみたらどうだ。」

発言部分は自動的にカギ括弧でくくられます。これを止めるには、設定ファイル「config-starter.yml」を変更します。またアイコン画像の大きさも変更できます。詳しくは後述します。

白黒印刷でも判別できる画像を使う

このサンプルでは、色が違うだけのアイコン画像を使っています。そのせいで、白黒印刷すると誰が誰だか判別できなくなります。もし白黒印刷するなら、色に頼らず判別できるようなアイコン画像を使いましょう。

アイコン画像を使わない場合

「//talk」の第1引数にアイコン画像を指定しない場合は、第2引数に名前を指定します。

▼ サンプル

```
//talklist{
//talk[][A准将]{
できもしないことをおっしゃらないください。
//}
//talk[][B提督]{
不可能なことを言い立てるのは貴官の方だ。
それも安全な場所から動かずにな。
//}
//talk[][A准将]{
……小官を侮辱なさるのですか。
//}
//talk[][B提督]{
貴官は自己の才能を示すのに、弁舌ではなく実績をもってすべきだろう。
他人に命令するようなことが自分にもできるかどうか、やってみたらどうだ。
//}
//}
```

▼ 表示結果

A 准将： 「できもしないことをおっしゃらないでください。」
B 提督： 「不可能なことを言い立てるのは貴官の方だ。それも安全な場所から動かずにな。」
A 准将： 「……小官を侮辱なさるのですか。」
B 提督： 「貴官は自己の才能を示すのに、弁舌ではなく実績をもってすべきだろう。他人に命令するようなことが自分にもできるかどうか、やってみたらどうだ。」

名前が長い場合は、会話の1行目だけ自動的に開始位置がずれます。

▼ サンプル

```
//talklist{  
//talk[][ビュコック提督]{  
  貴官は自己の才能を示すのに、弁舌ではなく実績をもってすべきだろう。  
  他人に命令するようなことが自分にもできるかどうか、やってみたらどうだ。  
//}  
//}
```

▼ 表示結果

ビュコック提督： 「貴官は自己の才能を示すのに、弁舌ではなく実績をもってすべきだろう。他人に命令するようなことが自分にもできるかどうか、やってみたらどうだ。」

アイコン画像と名前の両方を指定することは、現在はできません。できるようにして欲しい人は、相談してください。

コンパクトな書き方

ブロック本体のかわりに、「//talk」の第3引数に会話文を書けます。こうするとブロック本体を省略できるので、コンパクトな書き方になります。

また第3引数を使わない書き方と混在できます。文章が短い場合だけ第3引数を使い、ある程度の長さがある場合はブロック本体に書くのがいいでしょう。

▼ サンプル

```
//talklist{  
//talk[avatar-b][][できもしないことをおっしゃらないでください。]  
//talk[avatar-g][][不可能なことを言い立てるのは貴官の方だ。それも安全な場所から動かずにな。]  
//talk[avatar-b][][……小官を侮辱なさるのですか。]  
//talk[avatar-g]{  
  貴官は自己の才能を示すのに、弁舌ではなく実績をもってすべきだろう。
```

他人に命令するようなことが自分にもできるかどうか、やってみたらどうだ。

```
//}
```

```
//}
```

▼ 表示結果



「できないことをおっしゃらないでください。」



「不可能なことを言い立てるのは貴官の方だ。それも安全な場所から動かずにな。」



「……小官を侮辱なさるのですか。」



「貴官は自己の才能を示すのに、弁舌ではなく実績をもってすべきだろう。他人に命令するようなことが自分にもできるかどうか、やってみたらどうだ。」

会話文を第3引数に書く場合は、「`]`」を「`\]`」のように書いてください*32。そうしないとコンパイルエラーになるでしょう。

短縮名

大量の会話を入力する場合、いちいちアイコン画像や名前を入力するのは面倒です。そこで Starter では、会話に使うアイコン画像や名前を短縮名で登録できる機能を用意しました。

まず、「`config-starter.yml`」に短縮名を登録します。すでに以下のような設定がされているので、それを上書きするといいいでしょう。

▼ 「`config-starter.yml`」に短縮名を登録

```
talk_shortcuts:
  "b1":    {image: "avatar-b"} # アイコン画像を使う場合
  "g1":    {image: "avatar-g"} # アイコン画像を使う場合
  "#b1":   {name: "アリス"}    # アイコン画像を使わない場合
  "#g1":   {name: "ボブ"}      # アイコン画像を使わない場合
```

そして、原稿では「`//t[短縮名]`」のように書きます。たとえば「`//t[b1]`」は「`//talk[avatar-b]`」の短縮形となります。

前の項のサンプルを、短縮名を使って書いてみましょう。

*32 これは脚注を書くときと同じ制約です。

▼ サンプル

```
//talklist{  
//t[b1][できもしないことをおっしゃらないください。]  
//t[g1][不可能なことを言い立てるのは貴官の方だ。それも安全な場所から動かずにな。]  
//t[b1][……小官を侮辱なさるのですか。]  
//t[g1]{  
貴官は自己の才能を示すのに、弁舌ではなく実績をもってすべきだろう。  
他人に命令するようなことが自分にもできるかどうか、やってみたらどうだ。  
//}  
//}
```

▼ 表示結果



「できもしないことをおっしゃらないください。」



「不可能なことを言い立てるのは貴官の方だ。それも安全な場所から動かずにな。」



「……小官を侮辱なさるのですか。」



「貴官は自己の才能を示すのに、弁舌ではなく実績をもってすべきだろう。他人に命令するようなことが自分にもできるかどうか、やってみたらどうだ。」

短縮名を使うと、アイコン画像や登場人物名を変更するのも簡単になります。変更の発生が予想される場合も、短縮機能を使うといいでしょう。

カスタマイズ

「//talklist」の第1引数にオプションを指定すると、会話形式の表示をカスタマイズできます。たとえば次の例では、アイコン画像の幅を全角2文字分にし、会話文のインデント幅を全角4文字分にしています。

▼ サンプル

```
//talklist[imagewidth=2zw,indent=4zw]{  
//talk[avatar-g]{  
貴官は自己の才能を示すのに、弁舌ではなく実績をもってすべきだろう。  
他人に命令するようなことが自分にもできるかどうか、やってみたらどうだ。  
//}
```

//}

▼ 表示結果



「貴官は自己の才能を示すのに、弁舌ではなく実績をもってすべきだろう。他人に命令するようなことが自分にもできるかどうか、やってみたらどうだ。」

指定できるオプションは表 3.28 の通りです。これらは「`classname`」を除いて \LaTeX 用であり、HTML には「`classname`」しか適用されません。またオプションを指定できるのは「`//talklist`」だけであり、「`//talk`」には指定できません。

▼ 表 3.28: 「`//talklist`」のオプション

オプション名	デフォルト値	説明
<code>indent</code>	<code>6zw</code>	本文左端から会話部分までの幅
<code>separator</code>	<code>:</code>	名前と会話文との区切り文字列
<code>imagewidth</code>	<code>4zw</code>	アイコン画像の幅
<code>imageheight</code>		画像高さ（通常は無指定でよい）
<code>imageborder</code>	<code>off</code>	「 <code>on</code> 」なら画像に枠線をつける
<code>needvspace</code>	<code>4zw</code>	必要な空きがなければ改ページ
<code>itemmargin</code>	<code>1mm</code>	会話項目間の空き
<code>listmargin</code>	<code>4mm</code>	会話リストの最初と最後の空き
<code>itemstart</code>	<code>「</code>	会話文の先頭につける文字列
<code>itemend</code>	<code>」</code>	会話文の最後につける文字列
<code>classname</code>		HTML タグに含めるクラス名

表 3.28 のデフォルト値は、設定ファイル「`config-starter.yml`」で設定されています。必要に応じて変更してください。

▼ `config-starter.yml`

```
## 会話形式（`//talklist`）のデフォルトオプション
## （注：YAMLでは on は true と同じ、off は false と同じ）
talklist_default_options:
  indent:      6zw      # 本文左端から会話部分までの幅
  separator:   ":"      # 名前と会話文との区切り文字列
  imagewidth:  4zw      # アイコン画像の幅
  imageheight:      # 画像高さ（通常はnullでよい）
  imageborder: off      # on なら画像に枠線をつける
  needvspace:  4zw      # 必要な空きがなければ改ページ
  itemmargin:  1mm      # 会話項目間の空き
  listmargin:  0.5zw    # 会話リストの最初と最後の空き
  itemstart:   "「"     # 会話文の先頭につける文字列
  itemend:     "」"     # 会話文の最後につける文字列
```

```
classname:          # HTMLタグに含めるクラス名
```

たとえばアイコン画像の幅を全角3文字分にするには、次のように変更します。

▼ config-starter.yml

```
talklist_default_options:
  #...(省略)...
  imagewidth: 3zw    # アイコン画像の幅
  #...(省略)...
```

また会話文にカギ括弧をつけないようにするには、次のように変更します。

▼ config-starter.yml

```
talklist_default_options:
  #...(省略)...
  itemstart: ""      # 会話文の先頭につける文字列
  itemend:   ""      # 会話文の最後につける文字列
  #...(省略)...
```

なお会話形式に関する L^AT_EX マクロは「sty/starter-talk.sty」に定義されています。必要であれば、マクロを「sty/mystyle.sty」にコピーしてから変更してください。

HTML の場合は、「css/webstyle.css」をカスタマイズしてください。クラス名が「.talk-xxx」のエントリが会話形式に関するルールです。

▼ css/webstyle.css

```
.talk-image {
    width: 60px;    /* アイコン画像の表示幅 */
}
.talk-name {
    font-family: sans-serif; /* 名前のフォント */
    font-weight: normal;
}
.talk-text {
    display: block;
    margin-left: 100px; /* 会話文のインデント幅 */
    padding-top: 5px;
}
.... (省略) ....
```


3.17 用語、索引

たいていの本では、巻末に索引が用意されています（図 3.4）。Starter では、指定した用語を索引に登録できます。また用語に親子関係を持たせたり、転送先の用語を指定できます。

索引	
■ C	
C ボール	127
■ あ	
アルビオン	
王国	128
王立航空軍	128, 130
共和国	128
アンジェ	127
■ か	
空中艦隊	⇒ アルビオン王立航空軍
黒星紅白	127
ケイパーライト	128
症候群	128
■ さ	
ししおど	
鹿威し	127
侍従長	127
スパイ	129
二重——	129

▲ 図 3.4: 索引ページ

索引機能をオンにする

索引を作る機能は、デフォルトではオフになっています。オンにするには、`config.yml` の 370 行目ぐらいにある「`makeindex: true`」を設定してください。

なお索引に用語が何も登録されていない場合、索引機能をオンにすると PDF のコンパイルがエラーになります。1 つでもいいから本文に用語を指定してから、索引機能をオンにしてください。

ややこしいのですが、索引機能がオフでも用語を索引に登録できます。つまり、こういうことです：

- 索引機能がオンのとき、索引に用語が何も登録されていないとエラー。
- 索引機能がオフのとき、索引に用語を登録してもエラーにならない。

索引は \LaTeX の機能を使って生成しているので、索引機能は今のところ PDF でのみサポートしています。また索引機能をオンにすると、 \LaTeX のコンパイル回数が増えてコンパイルに時間がかかるようになります。執筆中は索引機能をオフにし、完成間近になったらオンにするといいでしょう。

用語の書き方

用語の書き方は次の通りです。

- 初めて登場した用語は「@<term>{用語名((よみかた))}」のように書きます。すると用語名がゴシック体で表示され、かつ巻末の索引に載ります。
- 再登場した用語は「@<idx>{用語名((よみかた))}」のように書きます。すると索引には載るけど、ゴシック体にはなりません。
- 「@<hid>{用語名((よみかた))}」(hidden index) とすると、索引に載るけど本文には何も表示されません。
- 用語名が漢字を含まない場合（つまりひらがなやカタカナや英数字だけの場合）は、読み方を省略して「@<term>{用語名}」のように書けます。
- 索引に登録せずゴシック体で表示したいだけのときは、「@<termnoid>{用語名}」と書きます。「@<hid>{ }」と組み合わせて使うことを想定しています。索引に登録しないので、よみがなは含めないでください。

▼ サンプル

```
@<term>{アンジェ((あんじえ))}。
@<idx>{侍従長((じじゅうちょう))}。
@<term>{Cボール}。
```

▼ 表示結果

アンジェ。侍従長。C ボール。

この例だと、「アンジェ」と「C ボール」は@<term>{ }を使っているのでゴシック体、「内務卿」は@<idx>{ }を使っているので本文と同じ明朝体のままです。

用語には他のインライン命令を含められます。その場合、必ずフェンス記法^{*33}を使って「@<term>|@<xxx>{...}|」のように書いてください（これは実装上の都合です）。「@<term>{@<xxx>{...}}」だとエラーになります。

▼ サンプル

```
@<term>|@<ruby>{鹿威, ししおど}し((ししおどし))|。
@<term>|@<ruby>{黒星, くろぼし}@<ruby>{紅白, こうはく}((くろぼしこうはく))|。
```

▼ 表示結果

ししおど 鹿威し。 くろぼしこうはく 黒星紅白。

本文では用語にルビをつけたいけど索引ではつけたくないなら、「@<hid>{ }」を使うといいでしょう。

^{*33} Re:VIEW では「@<foo>|...|」や「@<foo>\$...\$」のような書き方を「フェンス記法」と呼んでいます。

▼ サンプル

```
@<termnoidx>|@<ruby>{鹿威, ししおど}し|@<hidx>{鹿威し((ししおどし))}。
@<ruby>{黒星, くろぼし}@<ruby>{紅白, こうはく}@<hidx>{黒星紅白((くろぼしこうはく>
>))}。
```

親子関係にある用語

用語に親子関係がある場合、索引でそのように表示できます。そのためには用語を「<<>>」で区切ります。

たとえば2つの用語「ケイバーライト」と「ケイバーライト症候群」がある場合は、前者を親項目、後者を子項目にして索引に登録できます。また「アルビオン王国」「アルビオン共和国」「アルビオン王立航空軍」という3つの用語では、共通する「アルビオン」をあたかも存在する用語かのように見立てて索引に登録できます。

▼ サンプル

```
@<term>{ケイバーライト((けいばーらいと))}。
@<term>{ケイバーライト((けいばーらいと))<<>>症候群((しょうこうぐん))}。

@<term>{アルビオン((あるびおん))<<>>王国((おうこく))}。
@<term>{アルビオン((あるびおん))<<>>共和国((きょうわこく))}。
@<term>{アルビオン((あるびおん))<<>>王立航空軍((おうりつこうくうぐん))}。
```

▼ 表示結果

ケイバーライト。ケイバーライト症候群。
アルビオン王国。アルビオン共和国。アルビオン王立航空軍。

索引ページの表示結果は図 3.4 を見てください。これを見ると、「アルビオン」という用語は本文には登場していないので、索引の項目にページ番号がありません。これに対し、同じ親項目でも「ケイバーライト」は本文に登場したので、ページ番号がついています。

なお、「@<idx>{アルビオン<<>>王国}」と「@<idx>{アルビオン王国}」は別々の用語として扱われるので、索引にも別々に載ります。どちらかに統一しましょう。

また用語に「<<>>」を2つ埋め込むと、親・子・孫の関係を表せます（が推奨しません）。それ以上の深い関係は未対応です。

親子関係の順番を入れ替える

今までのサンプルでは、親項目が前にきて子項目がその後ろについていました（例：「アルビオン」＋「王国」）。そうではなく、親項目のほうが後ろ、子項目のほうが前にくる用語もあります。たとえば「二重スパイ」という用語では、後ろにある「スパイ」が親項目、前にある「二重」が子項目です。

- 「@<term>{二重((にじゅう))<<>スパイ((すぱい))}」だと、「二重」が親で「スパイ」が「子」になってしまいます。
- 「@<term>{スパイ((すぱい))<<>二重((にじゅう))}」だと、親子関係は正しいものの、本文には「スパイ二重」と表示されてしまいます。

つまり、どちらの書き方も正しくありません。

このような場合、子要素の用語名に「---」を含めると、そこに親要素が入るものとして扱います。たとえば「@<term>{スパイ((すぱい))<<>二重---((にじゅう))}」と書くと、「スパイ」が親で「二重」が子になる構造を指定しつつ、本文には「二重スパイ」と表示されます。

▼ サンプル

```
@<term>{スパイ((すぱい))}。
@<term>{スパイ((すぱい))<<>二重---((にじゅう))}。
```

▼ 表示結果

スパイ。二重スパイ。

索引ページの表示結果は図 3.4 を見てください。

念のために注意しますが、「---」を含めるのは子要素の用語名です。よみがなの中に入れてはいけません。また親要素に入れた場合は何の効果もなくそのまま表示されます。

転送先の用語を指定する

索引では、用語ごとにページ番号がつきます。ただし、ページ番号ではなく「～を見よ」のように転送先の用語を指定することがあります。このような場合は、「@<term>{用語((よみがな))==>>転送先}」のように書きます。たとえば「空中艦隊」が「アルビオン王立航空軍」の別名なら、次のようにして前者から後者へ転送するといいでしょう。

▼ サンプル

```
@<term>{アルビオン((あるびおん))<<>王立航空軍((おうりつこうくうぐん))}。
@<term>{空中艦隊((くうちゅうかんたい))==>>アルビオン王立航空軍}。
```

▼ 表示結果

アルビオン王立航空軍。空中艦隊。

実際に試すと、転送先を表すのに「→」が使われます。また表示結果(図 3.4)を見ると「→」に下線が引かれていますが、これは意図したものではありません。原因は調査中です。

なお、転送先の用語が索引に登録されているかどうかのチェックは行っていません。この場合だと、もし転送先の「アルビオン王立航空軍」が索引に登録されていなくてもエラーになりません。

これは仕様です*34。

よみがなの再利用

一度指定したよみがなは、その用語が再び登場したときに再利用されます。

たとえば、用語が初めて登場したときに「@<term>{侍従長 ((じじゅうちょう))}」のようによみがなを指定してます。そして用語が再び登場したときに「@<idx>{侍従長}」のようによみがなを省略すると、さきほどのよみがなが再利用されます。

親子関係にある用語でも、よみがなは再利用されます。たとえば「@@<nop><term>{アルビオン ((あるびおん))}<<>王国 ((おうこく))}」と一度書いておけば、その後は「@@<nop><idx>{アルビオン<<>王国}」のようによみがなを省略できます。

再登場した用語のよみがなは、省略してもいいししなくてもいいです。省略すれば前のよみがなが再利用され、省略しなければ再利用されないだけです。

よみがな辞書を使う

用語のよみがなは、辞書となるファイルに登録しておくこともできます。そうしておけば、用語のよみがなを省略できます。また複数人で執筆しているときに、索引に載せる用語と載せない用語を統一するのも辞書ファイルが役立ちます。

よみがなの辞書ファイルはタブ区切りのテキストファイルであり、`config.yml` の「`makeindex_dic: yomigana.txt`」(370 行目付近)でファイル名を指定します。デフォルトでは `yomigana.txt` というファイルが指定されており、その中身は用語とよみがなのリストです*35。用語とよみがなは1つ以上のタブ文字で区切ってください。

▼ よみがな辞書ファイルの中身

侍従長	じじゅうちょう
内務卿	ないむきょう
ハイランダー連隊	はいらんだーれんたい

こうしておけば、「@<term>{侍従長 ((じじゅうちょう))}」と書くかわりに「@<term>{じじゅうちょう}」のように読みを省略して書けます。

このほか、形態素解析エンジン「MeCab」を使うことでよみがなを用語から自動的に取得する機能がありますが、Starter では推奨していません。説明は省略するので、知りたい人は Re:VIEW のドキュメント*36を読んでください。

*34 索引の機能を L^AT_EX に任せているため、用語のチェックが難しいのです。

*35 このファイルは L^AT_EX において索引を生成する「mindex」というコマンドで読み込まれます。mindex コマンドで読み取れるのが、タブ区切りのテキストファイルなのです。CSV 形式は使えません。

*36 <https://github.com/kmuto/review/blob/master/doc/makeindex.ja.md>

用語と索引の補足事項

1 つの用語に複数の読み方がある場合

たとえば「一也」という用語の読み方が「かずや」と「いちや」の2つある場合は、面倒でも「@<idx>{一也 ((かずや))}」と「@<idx>{一也 ((いちや))}」を使い分けてください。

またこのようなケースがあるため、同じ用語に対して違うよみがなが指定されてもエラーにはなりません。

用語のよみがなが見つからなかった場合

よみがなが指定されず、よみがなの辞書ファイルにも登録されておらず、かつ漢字を含んだ用語（ひらがな・カタカナ・英数字以外の文字を含む用語）は、索引の最後に「■漢字」というグループがつくられ、そこに載ります。本に索引をつけた場合は、索引の最後に「■漢字」がないかチェックしましょう。もしあれば、よみがなを指定し忘れていました^{*37}。

索引の表示をカスタマイズする

索引の表示は、次のようなカスタマイズができます。

- 用語とページ番号の間を、連続したピリオドではなく空白にする。
- 用語のグループ化を行単位ではなく文字単位で行う。
- 用語見出しを「■あ」のようなデザインから変更する。
- 子要素の用語で使う「——」を別の文字列に変更する。
- 「@<term>{ }」の表示をゴシック体から別の書体に変更する。

詳しくは「4.7 索引」(p.162)を参照してください。なお索引のカスタマイズには、 $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ の知識が必要です。

Re:VIEW との違い

Starter の索引機能は、Re:VIEW のそれを拡張しています。違いは次の通りです。

- 「@<idx>{ }」と「@<hid>{ }」は、Re:VIEW にある機能です。Starter ではそれを引き継いでいます。
- 「@<term>{ }」は Starter による拡張であり、Re:VIEW にはありません。
- 「@<idx>{用語 ((よみがな))}」のようによみがなを埋め込む機能は、Starter による独自拡張です。
- 「@<idx>{親用語<<>>子用語}」のような親子関係の指定は、Re:VIEW にある機能です。Starter でも使えます。
- 「@<idx>{親<<>>子---}」のような親の位置を子で指定する機能は、Starter による独自拡張です。
- 「@<idx>{用語==>>転送先}」のような親子関係の指定は、Starter による独自拡張です。
- よみがな辞書ファイルの機能は、Re:VIEW にある機能です。

^{*37} よみがなの指定し忘れを警告にする機能は検討中です。

- 形態素解析エンジン「MeCab」を使ったよみがな自動取得機能は、Re:VIEW にある機能です。Starter でも設定すれば使えますが、トラブルが多いので推奨していません。

Re:VIEW の索引機能については、Re:VIEW のドキュメント^{*38}を参照してください。

.....

これを読んでも Re:VIEW 関係者の方へ

これを読んでも Re:VIEW 関係者の方へお願いがあります。Starter の機能を真似していただくのは構わないです（それどころか歓迎します）。しかし、特に理由がないのにわざと違う仕様にするのはユーザに不利益を与えるだけなので、止めていただくようお願いします。

Starter による拡張機能に問題があるから仕様を変えて導入した、というなら道理はあるでしょう。しかし Starter を新機能の参考にしておきながら、わざわざ違う仕様にして導入するのは、ユーザの利益になりません。

競うなら公平・公正に。ユーザの利益を犠牲にする必要はありません。

.....

^{*38} <https://github.com/kmuto/review/blob/master/doc/makeindex.ja.md>

3.18 単語展開

キーと単語の組を単語辞書ファイルに登録しておけば、キーを単語に展開できます。たとえば、辞書ファイル「data/words.txt」の中にキー「apple」と単語「アップル」を登録しておけば、「@<w>{apple}」が「アップル」に展開されます。

この機能を使うと、表記が定まっていない単語の表記をあとから簡単に変更できます。たとえば「Apple」と「アップル」のどちらの表記にするかまだ決まっていなくても、本文に「@<w>{apple}」と書いておけば、あとから辞書の中身を変えるだけでどちらの表記にも対応できます。

なお単語にはインライン命令を含められません。

単語展開用のインライン命令

単語展開のためのインライン命令は3つあります。

- 「@<w>{ }」は、キーに対応した単語へと展開するだけです。
- 「@<wb>{ }」は、展開してから太字にします。実質的に「@{@<w>{ } }」と同じです。
- 「@<W>{ }」は、展開してから強調表示します。実質的に「@{@<w>{ } }」と同じです。

▼ サンプル

```
* @<w>{apple} …… 展開するだけ
* @<wb>{apple} …… 展開して太字に
* @<W>{apple} …… 展開して強調表示
```

▼ 表示結果

- アップル …… 展開するだけ
- **アップル** …… 展開して太字に
- **アップル** …… 展開して強調表示

単語展開用の辞書ファイル

単語展開用の辞書ファイルは、config.yml の「words_file」で指定します。デフォルトでは「data/words.txt」が指定されています。

辞書ファイルは、CSV ファイル (*.csv) か、タブ文字区切りのファイル (*.txt または *.tsv) を指定します。どちらの形式かはファイル名の拡張子で判断します。

辞書ファイルは複数指定できます。たとえば「words_file: [data/f1.txt, data/f2.txt]」とすれば、data/f1.txt と data/f2.txt を読み込みます。キーが重複していてもエラーにはならず、あとから読み込まれたほうが使われます。

辞書ファイルの文字コードは UTF-8 しか使えません。

Re:VIEW との違い

- Re:VIEW では、辞書ファイルとして CSV ファイルしか指定できません。Starter だと、タブ文字区切りのテキストファイルも指定できます。テキストエディタで編集しやすいのは、CSV ファイルよりもタブ文字区切りのテキストファイルのほうです。
- Re:VIEW には「@<w>」がありません。
- Starter ではデフォルトで辞書ファイル「data/words.txt」が提供済みです。設定ファイルを編集しなくても最初から@<w>{ }が利用可能です。

3.19 その他

URL、リンク

URL は「@<href>{URL}」と書きます。

▼ サンプル

```
Re:VIEW Starter :@<href>{https://kauplan.org/reviewstarter/}
```

▼ 表示結果

Re:VIEW Starter : <https://kauplan.org/reviewstarter/>

リンクは「@<href>{URL, テキスト}」と書きます。

▼ サンプル

```
@<href>{https://kauplan.org/reviewstarter/, Re:VIEW Starter}
```

▼ 表示結果

Re:VIEW Starter^{*39}

PDF の場合、リンクの URL は自動的に脚注に書かれます (Starter 拡張)。これは印刷時でも URL が読めるようにするためです。この拡張機能をオフにするには、`starter-config.yml` で設定項目「`linkurl_footnote:`」を「`false`」または「`off`」にします。

もし「`linkurl_footnote:`」の設定に関係なく、必ずリンクを使いたい (URL を脚注に書かない) 場合は、「@<hlink>{...}」を使ってください。使い方は「@<href>{...}」と同じです。

「@<href>{ }」と「@<hlink>{ }」の違いを比べてみましょう。

▼ サンプル

```
* @<href>{https://twitter.com/_kauplan/, @_kauplan}
* @<hlink>{https://twitter.com/_kauplan/, @_kauplan}
```

▼ 表示結果

- @_kauplan^{*40}
- @_kauplan

^{*39} <https://kauplan.org/reviewstarter/>

^{*40} https://twitter.com/_kauplan/

引用

引用は「`//quote{ ... //}`」で表します。

▼ サンプル

```
//quote{
//noindent
その者、蒼き衣を纏いて金色の野に降り立つべし。@<br>{}
失われし大地との絆を結び、ついに人々を青き清浄の地に導かん。
//}
```

▼ 表示結果

その者、蒼き衣を纏いて金色の野に降り立つべし。
失われし大地との絆を結び、ついに人々を青き清浄の地に導かん。

傍点

傍点は「`@<bou>{...}`」でつけられます。

▼ サンプル

```
@<bou>{三分間}、待ってやる。
```

▼ 表示結果

三分間、待ってやる。

傍点は、太字で強調するほどではないけど読者の注意を向けたいときに使います。ただしアルファベットや記号ではあまりよいデザインにはなりません。

ルビ、読みがな

ルビ（あるいは読みがな）は「`@<ruby>{テキスト, ルビ}`」のように書きます。

▼ サンプル

```
@<ruby>{約束された勝利の剣, エクスカリバー}
```

▼ 表示結果

エクスカリバー
約束された勝利の剣

途中の「, 」は必ず半角文字を使い、全角文字は使わないでください。

何もしないコマンド

「@<nop>{ }」^{*41}は、何もしないコマンドです。これは、たとえばインライン命令の「@」と「<」の間に入れることで、インライン命令として解釈されないようごまかすために使います。

▼ サンプル

強調される： @{テキスト}。

強調されない： @@<nop>{ }{テキスト}。

▼ 表示結果

強調される： **テキスト**。

強調されない： @{テキスト}。

またブロック命令の行頭に「@<nop>{ }」を入れると、ブロック命令として解釈されなくなります。

コード中コメント

プログラムコードやターミナルにおいて、コメントを記述するための「@<balloon>{...}」というコマンドが用意されています。

▼ サンプル

```
//terminal{
$ @<userinput>{unzip mybook.zip}      @<balloon>{zipファイルを解凍}
$ @<userinput>{cd mybook/}             @<balloon>{ディレクトリを移動}
$ @<userinput>{rake pdf}               @<balloon>{PDFファイルを生成}
//}
```

▼ 表示結果

```
$ unzip mybook.zip      ← zipファイルを解凍
$ cd mybook/           ← ディレクトリを移動
$ rake pdf              ← PDFファイルを生成
```

Re:VIEW のドキュメントを読むと、本来は吹き出し（バルーン）形式で表示することを意図していたようです。Starter では、「←」つきのグレーで表示しています。

なおプログラムコードにおいて改行記号を表示している場合、「@<balloon>{ }」があるとそれより後に改行文字が表示されてしまいます。たとえば次の例だと、改行記号は「←再帰呼び出し」

^{*41} 「nop」は「No Operation」の略です。

の前につくべきですが、実際には後についています。これは期待される挙動ではないでしょう。

▼ サンプル

```
//program[] [改行記号と「@@<nop>{}<balloon>{}」は相性が悪い] [eolmark=on]{
function fib(n) {
  if (n <= 1) { return n };
  return fib(n-1) + fib(n-2);  @<balloon>{再帰呼び出し}
}
//}
```

▼ 表示結果

▼ 改行記号と「@<balloon>{}」は相性が悪い

```
function fib(n) {
  if (n <= 1) { return n };
  return fib(n-1) + fib(n-2);  ←再帰呼び出し
}
```

この問題については将来的に対応予定ですが、現在はまだ対応できていません。

特殊文字

- 「@<hearts>{}」で「♡」が表示できます。
- 「@<TeX>{}」で「 $\mathrm{T}_\mathrm{E}\mathrm{X}$ 」が表示できます。
- 「@<LaTeX>{}」で「 $\mathrm{L}^{\mathrm{A}}\mathrm{T}_{\mathrm{E}}\mathrm{X}$ 」が表示できます。

右寄せ、センタリング

右寄せとセンタリングのブロック命令があります。

▼ サンプル

```
//flushright{
右寄せのサンプル
//}
//centering{
センタリングのサンプル
//}
```

▼ 表示結果

右寄せのサンプル

センタリングのサンプル

縦方向の空き（実験的機能）

「`//vspace`」を使って、縦方向の空きを入れられます。マイナスの値を指定すると、空きを削除できます。

▼ サンプル

```
AAAAA  
  
BBBBB  
  
//vspace[latex][-4mm]  
CCCCC
```

▼ 表示結果

AAAAA
BBBBB

生データ

L^AT_EX のコード（PDF のとき）や HTML のコード（ePub のとき）を埋め込む機能があります。たとえば次の例では、PDF のときは「`//embed[latex]{...//}`」のコードが使われ、HTML と ePub のときだけ「`//embed[html,epub]{...//}`」のコードが使われます。

▼ サンプル

```
//embed[latex]{  
\textcolor{red}{Red}  
\textcolor{green}{Green}  
\textcolor{blue}{Blue}  
\par  
//}  
  
//embed[html,epub]{  
<div>  
  <span style="color:red">Red</span>  
  <span style="color:green">Green</span>  
  <span style="color:blue">Blue</span>  
</div>  
//}
```

▼ 表示結果

Red Green Blue

またインライン命令の「`@<embed>{...}`」もあります。

- 「`@<embed>{|latex|...}`」なら \LaTeX のときだけ中身が出力されます。
- 「`@<embed>{|html,epub|...}`」なら HTML と ePub のときだけ中身が出力されます。

参考文献

(未執筆。Re:VIEW の Wiki^{*42}を参照してください。)

^{*42} <https://github.com/kmuto/review/blob/master/doc/format.ja.md#%E5%8F%82%E8%80%83%E6%96%87%E7%8C%AE%E3%81%AE%E5%AE%9A%E7%BE%A9>

第 4 章

カスタマイズ

Starter をカスタマイズする方法を紹介します。

4.1 命令

新しい命令を追加するには、ファイル「review-ext.rb」を編集します。

新しいインライン命令を追加する

たとえば次の例では、新しいインライン命令「@<bold>{...}」を追加しています。

▼ ファイル「review-ext.rb」

```
module ReVIEW

  ## インライン命令「@<bold>{ }」を宣言
  Compiler.define_inline :bold

  ## LaTeX用の定義
  class LATEXBuilder
    def on_inline_bold(&b) # 入れ子を許す場合
      return "\\textbf{#{yield}}"
    end
    #def inline_bold(str) # 入れ子をさせない場合
    # return "\\textbf{#{escape(str)}}"
    #end
  end

  ## HTML (ePub) 用の定義
  class HTMLBuilder
    def on_inline_bold(&b) # 入れ子を許す場合
      return "<b>#{yield}</b>"
    end
  end
```

```

    #def inline_bold(str)    # 入れ子をさせない場合
    #   return "<b>#{escape(str)}</b>"
    #end
  end

end

```

なお既存のインライン命令を変更するのも、これと同じ方法で行えます。

新しいブロック命令を追加する

たとえば次の例では、新しいブロック命令「`//blockquote{ ... //}`」を追加しています^{*1}。

▼ ファイル「review-ext.rb」

```

module ReVIEW

  ## ブロック命令「//blockquote」を宣言（引数は0～1個）
  Compiler.defblock :blockquote, 0..1

  ## LaTeX用の定義
  class LATEXBuilder
    ## 入れ子を許す場合
    def on_blockquote_block(arg=nil)
      if arg.present?
        puts "\\noindent"
        puts escape(arg)
      end
      puts "\\begin{quotation}"
      yield
      puts "\\end{quotation}"
    end
    ## 入れ子を許さない場合
    def blockquote(lines, arg=nil)
      # if arg.present?
      #   puts "\\noindent"
      #   puts escape(arg)
      # end
      # puts "\\begin{quotation}"
      # puts lines
      # puts "\\end{quotation}"
    end
  end
end

```

^{*1} インライン命令のときは「`on_inline_xxx()`」なのにブロック命令では「`on_xxx_block()`」なのは、歴史的な事情です。非対称で格好悪いですが、ご了承ください。

```

end

## HTML (ePub) 用の定義
class HTMLBuilder
  ## 入れ子を許す場合
  def on_blockquote_block(arg=nil)
    if arg.present?
      puts "<p><span>#{escape(arg)}</span></p>"
    end
    puts "<blockquote>"
    yield
    puts "</blockquote>"
  end
  ## 入れ子を許さない場合
  #def blockquote(lines, arg=nil)
  #  if arg.present?
  #    puts "<p><span>#{escape(arg)}</span></p>"
  #  end
  #  puts "<blockquote>"
  #  puts lines
  #  puts "</blockquote>"
  #end
end
end

```

ブロック命令には「{ ... //}」を使わないタイプのものもあります。その場合は「Compiler.defsingle」を使います。

たとえば縦方向にスペースを空ける「//verticalspace[タイプ][高さ]」というブロック命令を追加するとしましょう。使い方は次の通りとします。

```

//verticalspace[latex][5mm]  ← PDFのときは5mm空ける
//verticalspace[epub][10mm]  ← ePubのときは10mm空ける
//verticalspace[*][15mm]     ← PDFでもePubでも15mm空ける

```

このようなブロック命令は、次のように定義します。「Compiler.defblock」ではなく「Compiler.defsingle」を使っている点に注意してください。

▼ ファイル「review-ext.rb」

```

module ReVIEW

  ## ブロック命令「//verticalspace」を宣言（引数は2個）
  Compiler.defsingle :verticalspace, 2

```

```
## LaTeX用の定義
class LATEXBuilder
  ## '{ ... //}' を使わないので入れ子対応にはしない
  def verticalspace(type, length)
    if type == 'latex' || type == '*'
      puts "\\vspace{#{length}}"
    end
  end
end

## HTML (ePub) 用の定義
class HTMLBuilder
  ## '{ ... //}' を使わないので入れ子対応にはしない
  def veritcalspace(type, length)
    if type == 'html' || type == 'epub' || type == '*'
      puts "<div style=\"height: #{length}\"></div>"
    end
  end
end

end
```

なお既存のブロック命令を変更するのも、これらと同じ方法で行えます。

4.2 ページと本文

[PDF] ページサイズを B5 や A5 に変更する

ページサイズを B5 や A5 に変更するには、「config-starter.yml」の設定項目「pagesize:」を変更します*2。

▼ ファイル「config-starter.yml」

```
pagesize: A5    # A5 or B5
```

[PDF] 本文の幅を指定する

本文の幅を指定するには、「config-starter.yml」の設定項目「textwidth:」を変更します*3。

▼ ファイル「config-starter.yml」

```
textwidth: 39zw    # 全角文字数で指定
```

ここで、「zw」は全角 1 文字分の幅を表す単位です。「39zw」なら全角 39 文字分の幅（長さ）を表します。

Starter では表 4.1 のような本文幅を推奨しています。

▼ 表 4.1: 本文幅の推奨値（タブレット向けを除く）

ページサイズ	フォントサイズ	本文幅
A5	9pt	全角 38 文字か 39 文字
A5	10pt	全角 35 文字
B5	10pt	全角 44 文字か 45 文字
B5	11pt	全角 41 文字

Starter では、印刷用 PDF ではページ左右の余白幅を変えています。これは印刷時の読みやすさを確保したまま、本文の幅を最大限に広げているからです。詳しくは「[PDF] 印刷用では左右の余白幅を充分にとる」(p.206) をご覧ください。

[PDF] 本文の高さを指定する

本文の高さを調整するには、「sty/mytextsize.sty」を編集します。

▼ ファイル「sty/mytextsize.sty」

```
% 天（ページ上部）の余白を狭め、その分を本文の高さに加える
\addtolength{\topmargin}{-2\Cvs} % 上部の余白を2行分減らす
```

*2 以前は「config.yml」の中の「texdocumentclass:」を変更していましたが、その必要はなくなりました。

*3 以前は「sty/mytextsize.sty」を編集していましたが、本文幅を変更するだけならその必要はなくなりました。

```
\addtolength{\textheight}{2\Cvs} % 本文の高さを2行分増やす
```

通常は、このような変更は必要ないはずです。

[PDF] 塗り足しをつける

印刷所に入稿するとき、本文の全ページに上下左右 3mm の「塗り足し」が必要になることがあります。塗り足しについては次のページなどを参考にしてください。

- 原稿作成の基礎知識：塗り足し（同人誌印刷栄光）
<https://www.eikou.com/making/basis#nuritashi>
- 原稿作成マニュアル：印刷範囲と塗り足し（同人誌印刷の緑陽社）
<https://www.ryokuyou.co.jp/doujin/manual/basic.html#nuritashi>

塗り足しは、技術系同人誌の本文ではたいていの場合必要ありませんが、次のような場合は必要です。

- ページ端まで絵や線が描かれている場合
- 背景色をつけたページがある場合（大扉や章扉など）

技術系同人誌でも表紙と裏表紙には塗り足しが必要

同人誌作成の経験がない方のために補足説明をします。

- 印刷所に入稿するとき、表紙や裏表紙と、本文とは別の PDF ファイルにします。前者はカラー印刷、後者は白黒印刷です。
- 表紙や裏表紙では背景色をつけることがほとんどなので、塗り足しが必要です。
- マンガ系の同人誌では背景色があったりページ端まで絵を描くことがあるので、塗り足しが必要です。同人誌印刷所のサイトで「塗り足しは必須です」と説明されているのは、マンガ同人誌を前提としているからです。
- 技術系同人誌の本文では背景色をつけないしページ端まで何かを描くこともないので、塗り足しは必要ありません。ただしこれは印刷所によって違う可能性があるので、詳しくは入稿先の印刷所に聞いてみてください。
- 塗り足しが必要なのに塗り足しをつけてしまっても、問題ありません。
- 塗り足しは印刷するときのみ必要であり、印刷しないなら必要ありません。

Starter では、塗り足しをつける設定が用意されています。

▼ config-starter.yml：塗り足しをつける

```
## 上下左右の塗り足し幅（印刷用PDFのみ）
bleedsizes: 3mm      # 上下左右に3mmの塗り足しをつける
```

塗り足しがつくのは印刷用 PDF のみであり、電子用にはつきません^{*4}。また塗り足しは上下左

^{*4} 印刷用 PDF と電子用 PDF の違いについては、「2.5 印刷用 PDF と電子用 PDF」(p.33) を参照してください。

右につくので、塗り足し幅が 3mm だとページサイズの縦と横は 6mm ずつ増えます。

[PDF] 塗り足しを可視化する

塗り足しがどのくらいの幅を取るのか、見た目で確認したい人もいるでしょう。そのような場合は、`config-starter.yml` で「`bleedcolor: blue`」のように指定してください。すると塗り足しが可視化されます。色は `red/green/blue/yello/gray` などが使えます。

この機能は、あくまで確認用のみで使ってください。入稿するときは必ず設定を外してください。

[PDF] 章の右ページ始まりをやめる

横書きの本では、章 (Chapter) を見開きの右ページから始めるのが一般的です。そのため、その前のページが空ページになることがよくあります。この空ページを入れたくないなら、章の右ページ始まりを止めて左右どちらのページからも始めるようにします。

右ページ始まりをやめるには、「`config.yml`」の設定項目「`texdocumentclass:`」を変更します。

▼ ファイル「`config.yml`」

```
texdocumentclass: ["jsbook",
  "dvipdfmx,uplatex,papersize,twoside,openright" ←右ページはじまり
  "dvipdfmx,uplatex,papersize,twoside,openany"   ←両ページはじまり
]
```

[PDF] 「`//info`」や「`//warning`」のデザインを変更する

Re:VIEW や Starter では、「`//note`」以外にも「`//info`」や「`//warning`」が使えます（詳しくは「ノート以外のミニブロック」(p.85) を見てください)。

「`//info`」や「`//warning`」で使われているアイコン画像を変更するには、「`sty/mystyle.sty`」にたとえば次のように書いてください。

▼ ファイル「`sty/mystyle.sty`」

```
\def\starter@miniblock@info@imagefile{./images/info-icon.png}
\def\starter@miniblock@warning@imagefile{./images/warning-icon.png}
```

アイコン画像の表示サイズや本文のフォントを変更するには、次のように書いてください。

▼ ファイル「`sty/mystyle.sty`」

```
\def\starter@miniblock@imagewidth{3zw} % アイコン画像の表示幅
\def\starter@miniblock@indent{4zw}     % 本文のインデント幅
\def\starter@miniblock@font{\small}    % 本文のフォント
```

「`//memo`」や「`//tips`」や「`//important`」などには、デフォルトではアイコン画像はつきません。これらにもアイコン画像をつけるには、たとえば次のように書いてください。

▼ ファイル「sty/mystyle.sty」

```
% 「//important」にアイコン画像をつける
\newenvironment{starterimportant}[1]{%
  \begin{starter@miniblock}[\starter@miniblock@important@imagefile]{#1}%
}{%
  \end{starter@miniblock}%
}
\def\starter@miniblock@important@imagefile{./images/info-icon.png}
```

デザインそのものを変更するには、たとえば次のようにしてください（どんな表示になるかは自分で確かめてください）。

▼ ファイル「sty/mystyle.sty」

```
% 「//memo」のデザインを変更する
\usepackage{ascmac}
\renewenvironment{startermemo}[1]{%
  \addvspace{1.5\baselineskip}% 1.5行分空ける
  \begin{boxnote}%           % 飾り枠で囲む
    \ifempty{#1}\else%       % 引数（タイトル）があれば
      {\centering%           % センタリングして
        \headfont\Large#1\par}% 太字のゴシック体で大きく表示
      \bigskip%               % 縦方向にスペースを空ける
    \fi%
    \setlength{\parindent}{1zw}% 段落の先頭を字下げする
  }{%
    \end{boxnote}%           % 飾り枠の終わり
    \addvspace{1.5\baselineskip}% 1.5行分空ける
  }
```


4.3 フォント

[PDF] フォントサイズを変更する

本文のフォントのサイズを変更するには、「config-starter.yml」の設定項目「fontsize:」を変更します*5。

▼ ファイル「config-starter.yml」

```
fontsize: 9pt      # 9pt or 10pt
```

フォントサイズは、A5 サイズなら 9pt、B5 サイズなら 10pt にするのが一般的です。ただし初心者向けの入門書では少し大きくして、A5 サイズなら 10pt、B5 サイズなら 11pt にするのがいいでしょう。

..... 商業誌のフォントサイズと本文幅

A5 サイズの商業誌で使われているフォントサイズと本文幅を調べました (表 4.2)。参考にしてください。

▼ 表 4.2: 商業誌のフォントサイズと本文幅

出版社	書籍名	各種サイズ
オライリー・ジャパン	『リーダブルコード』	A5、9pt、38 文字
	『ゼロから作る Deep Learning』	A5、9pt、38 文字
翔泳社	『達人に学ぶ SQL 徹底指南書 第 2 版』	A5、9pt、38 文字
オーム社	『アジャイル・サムライ』	A5、10pt、36 文字
技術評論社	『オブジェクト指向 UI デザイン』	A5、9pt、35 文字
	『現場で役立つシステム設計の原則』	A5、10pt、33 文字
日経 BP 社	『独習プログラマー』	A5、10pt、34 文字

.....

[PDF] フォントの種類を変更する

本文のフォントを変更するには、「config-starter.yml」の設定項目「fontfamily_ja:」と「fontfamily_en:」を変更します。

▼ ファイル「config-starter.yml」

```
## 本文のフォント（注：実験的）
## （Notoフォントが前提なので、Docker環境が必要。MacTeXでは使わないこと）
fontfamily_ja: mincho  ← mincho: 明朝体, gothic: ゴシック体
fontfamily_en: roman   ← roman: ローマン体, sansserif: サンセリフ体
```

*5 以前は「config.yml」の設定項目「texdocumentclass:」を変更していましたが、その必要はなくなりました。

```
fontweight_ja: normal    ← normal: 通常, light: 細字  
fontweight_en: normal    ← normal: 通常, light: 細字
```

または、「sty/mystyle.sty」に以下を追加します。

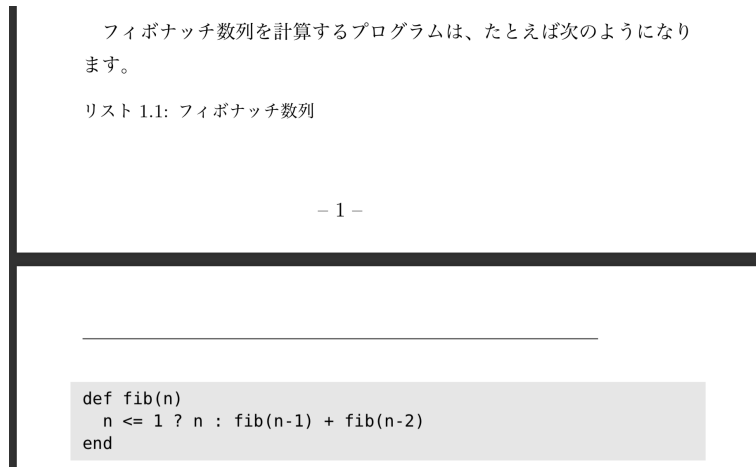
▼ ファイル「sty/mystyle.sty」

```
%% 英数字のフォントをサンセリフ体に変更  
\renewcommand\familydefault{\sfdefault}  
%% 日本語フォントをゴシック体に変更  
\renewcommand\kanjifamilydefault{\gtdefault}  
%% (macOSのみ) 日本語フォントを丸ゴシックに変更  
\renewcommand\kanjifamilydefault{\mgdefault}
```

4.4 プログラムコード

[PDF] 説明文直後での改ページを防ぐ

プログラムコードの説明文の直後で改ページされてしまうことがあります。すると説明文とプログラムコードが別ページに離れてしまうため、読みにくくなります（図 4.1）。



▲ 図 4.1: 説明文の直後で改ページされることがある

Starter ではこれを防ぐための工夫をしていますが、それでも発生してしまうことがあります。もし発生したときは、次の対策のうちどちらを行ってください。

個別対策：「`//needvspace[]`」を使う

たとえば「`//needvspace[latex][6zw]`」とすると、現在位置に全角 6 文字分の高さのスペースがあるかを調べ、なければ改ページします。

▼ 全角 6 文字分の高さのスペースがなければ改行

```
//needvspace[latex][6zw]
//list[][フィボナッチ数列]{
def fib(n)
  n <= 1 ? n : fib(n-1) + fib(n-2)
end
//}
```

全体対策：「`caption_needspace:`」の値を増やす

`config-starter.yml` の「`caption_needspace:`」がたとえば「`4.0zh`」であれば、プログラムリストの説明文を表示する前に全角 4 文字分の高さのスペースがあるかを調べ、なければ改ページします。よってこの設定値を増やせば、説明文だけが別ページになるのを防げます。

ただしこの設定値は、説明文を持つすべてのプログラムリストに影響します。設定値を変える場

合は注意してください。

[PDF] プログラムコードのフォントを変更する

プログラムコードやターミナルのフォントを変更するには、「config-starter.yml」の設定項目「program_ttfont:」と「terminal_ttfont:」を変更します。

▼ ファイル「config-starter.yml」

```
## プログラム用 (//list) の等幅フォント
program_ttfont: beramono
## ターミナル用 (//terminal、//cmd) の等幅フォント
terminal_ttfont: inconsolata-narrow
```

どんなフォントが使えるかは、「config-starter.yml」の中に書いてあります。

プログラムコードの枠線

プログラムコードに枠線をつける・つけないを変更するには、次の設定を変更してください。

▼ ファイル「config-starter.yml」

```
## プログラム (//list) を枠で囲むならtrue、囲まないならfalse
program_border: true
```

プログラムコードがページをまたぐときは、枠線付きのほうが分かりやすいです。詳しくは「ページまたぎしていることを可視化する」(p.201) を見てください。

オプションのデフォルト値

「//list」の第3引数には、さまざまなオプションが指定できます。これらのオプションは、デフォルト値が「config-starter.yml」で次のように設定されています。ここで、YAML^{*6}では「on, off, ~」がそれぞれ「true, false, null」と同じ意味になります。

▼ config-starter.yml

```
## プログラム (//list) のデフォルトオプション
## (注:YAMLでは `on, off, ~` は `true, false, null` と同じ)
program_default_options:
  fold:      on      # on なら長い行を右端で折り返す
  foldmark:  on      # on なら折り返し記号をつける
  eolmark:   off     # on なら行末に改行記号をつける
  lineno:    ~       # 1 なら行番号をつける
  linenewidth: ~     # 行番号の幅を半角文字数で指定する
```

^{*6} 「YAML」とはデータを構造化して記述する書き方のひとつであり、JSONの親戚みたいなものです。config-starter.ymlはYAMLで書かれています。

```

indent:      ~      # インデント幅
fontsize:    ~      # small/x-small/xx-small
lang:        ~      # デフォルトのプログラミング言語名
#file:       ~      # （使用しない）
encoding:    utf-8   # ファイルの文字コード

```

たとえば「`//list`」にデフォルトで行番号とインデント記号をつけるには、設定を次のように変更します。

▼ 「`//list`」にデフォルトで行番号とインデント記号をつける

```

program_default_options:
  #...省略...
  lineno:      1      # 1 から始まる行番号をつける
  linenewidth: 2      # 行番号を2桁で（0なら自動計算）
  indent:      4      # インデント幅を半角空白4文字に
  #...省略...

```

このようにデフォルト値を変更すれば、「`//list`」にオプションを何も指定しなくても、行番号とインデント記号がつくようになります。

なおこのように設定をしたうえで、特定のプログラムコードでだけ行番号やインデント記号をつけたくない場合は、「`//list[][][lineno=off,indent=off]`」のように個別の指定をしてください。

またターミナル画面を表す「`//terminal`」のオプションも、デフォルト値が `config-starter.yml` で設定されています。項目名が「`terminal_default_options:`」であること以外はプログラムコードの場合と同じです。

[PDF] 折り返し記号や行番号をマウス選択の対象から外す

PDF 中のプログラムコードをマウスで選択してコピーすると、折り返し記号や行番号や改行記号もコピーされてしまいます。これはプログラムコードをコピーするときに不便です。

Starter では、これらの記号をマウスの選択対象から外すことができます。そのためには、次のどちらかを設定してください。

- `config-starter.yml` の「`exclude_mouseselect:`」を `true` にする。
- 環境変数「`$STARTER_EXCLUDE_MOUSESELECT`」を「`on`」に設定する（「`true`」ではダメ）。

この設定をしてから PDF を生成すると、以下のものがマウスの選択対象にならなくなります^{*7}。

- 折り返し記号
- 行番号

^{*7} 「`@<balloon>{}`」で埋め込んだコメント文字列は、マウスコピーの対象外ではありません。これは意図した仕様です。もしこれも対象外にしたい場合は、相談してください。

- 改行記号

なおインデントを表す記号は、文字ではなく図形として縦線を描写しているため、マウスでの選択対象にはなりません。

ただし、この機能が有効なのは Acrobat Reader のみです。macOS の Preview.app では効かないので注意してください。

またこの設定をオンにすると、 \LaTeX のコンパイル時間が大幅に増えます。そのため、デフォルトではオフになっています。リリース時にだけ環境変数「`$STARTER_EXCLUDE_MOUSESELECT`」を on にするといいいでしょう。

▼ リスト 4.1: 「`$STARTER_EXCLUDE_MOUSESELECT`」を on にして PDF を生成

```
$ STARTER_EXCLUDE_MOUSESELECT=on rake pdf
```

4.5 見出し

章のタイトルデザインを変更する

章 (Chapter) のタイトルデザインを変更するには、「config-starter.yml」の設定項目「chapter_decoration:」を変更します。

▼ ファイル「config-starter.yml」

```
## 章 (Chapter) タイトルの装飾
## (none: なし、underline: 下線、boldlines: 上下に太線)
chapter_decoration: boldlines

## 章 (Chapter) タイトルの左寄せ/右寄せ/中央揃え
## (left: 左寄せ、right: 右寄せ、center: 中央揃え)
chapter_align: center

## 章 (Chapter) タイトルを1行にする (章番号とタイトルとの間で改行しない)
chapter_online: false
```

節のタイトルデザインを変更する

節 (Section) のタイトルデザインを変更するには、「config-starter.yml」の設定項目「section_decoration:」を変更します。

▼ ファイル「config-starter.yml」

```
## 節 (Section) タイトルの装飾
section_decoration: grayback
```

どんなデザインが指定できるかは、図 4.2 を参照してください。

.....

節タイトルが長くて 2 行になる場合

もし節タイトルが長くて 2 行になることがある場合は、「circle」「leftline」「numbox」のどれかを指定するといでしょう。それ以外だと、デザインが破綻してしまいます。

.....

段見出しのデザインを変更する

段 (Paragraph) 見出しや小段 (Paragraph) 見出しのデザインを変更するには、「sty/starter-heading.sty」から「\Paragraph@title」や「\Subparagraph@title」の定義を取り出し、「sty/mystyle.sty」にコピーしたうえで変更してください。

「`section_decoration: grayback`」の場合：

1.1 節タイトル：grayback

「`section_decoration: numbox`」の場合：

1.1 節タイトル：numbox

「`section_decoration: underline`」の場合：

1.1 節タイトル：underline

「`section_decoration: leftline`」の場合：

1.1 節タイトル：leftline

「`section_decoration: circle`」の場合：

1.1 節タイトル：circle

▲ 図 4.2: 節タイトルのデザイン

[PDF] 章や節のタイトルを独自にデザインする

章 (Chapter) や節 (Section) のタイトルのデザインは、「sty/starter-heading.sty」で定義されています。これらを変更したい場合は、必要な箇所を「sty/mystyle.sty」にコピーしてから変更してください。

またマクロを上書きする場合は、「\newcommand」ではなく「\renewcommand」を使うことに注意してください。

[PDF] 章扉をつける

Starter では、章 (Chapter) ごとにタイトルページをつけられます (図 4.3)。これを「章扉」といい、商業誌ではとてもよく見かけます。



▲ 図 4.3: 章扉 (章の概要と目次つき)

章扉をつけるには、次のようにします。

- 章タイトル直後に「`//abstract{ ... //}`」で概要を書く (必須)。
- そのあとに「`//makechaptitlepage[toc=on]`」を書く。
- これをすべての章に対して行う (まえがきやあとがきは除く)。

▼ 章扉をつける

```
= 章タイトル

//abstract{
...章の概要...
//}

//makechaptitlepage[toc=on]

== 節タイトル
```

章扉では、その章に含まれる節 (Section) の目次が自動的につきます。これは読者にとって便利なので、特にページ数の多い本では章ごとのタイトルページをつけることをお勧めします。

[PDF] 章扉に背景色をつける

章扉には、デフォルトでは背景色がついていませんが、背景色をつけると見栄えがかなりよくなります。

背景色は `sty/config-color.sty` において「`starter@chaptitlepagecolor`」という名前で定義されているので、これを上書き設定すれば変更できます。設定は `sty/mystyle.sty` で行うといいでしょう。

▼ `sty/mystyle.sty` : 章扉の背景色を設定する

```
\definecolor{starter@chaptitlepagecolor}{gray}{0.9}% 明るいグレー
```

また章扉に背景色を設定した場合は、印刷用 PDF の上下左右に 3mm ずつの「塗り足し」をつけるのが望ましいです。そのためには、`config-starter.sty` で「`bleedsizesize: 3mm`」を設定してください。この設定は印刷用 PDF でのみ機能し、電子用 PDF では無視されます。詳しくは「[PDF] 塗り足しをつける」(p.142) を参照してください。

[PDF] 節ごとに改ページする

初心者向けの入門書では、節 (Chapter) ごとに改ページするデザインが好まれます。Starter でこれを行うには、「`config-starter.yml`」で設定項目「`section_newpage:`」を「`true`」に設定します。

▼ ファイル「`config-starter.yml`」

```
## 節 (Section) ごとに改ページするか
## (ただし各章の最初の節は改ページしない)
section_newpage: true
```

項に番号をつける

Re:VIEW や Starter では、デフォルトでは項 (Subsection) に番号がつきません。項にも番号をつけるには、「config.yml」の設定項目「secnolevel:」を「3」に変更します。

▼ ファイル「config.yml」

```
# 本文でセクション番号を表示する見出しレベル
#secnolevel: 2    ←章 (Chapter) と節 (Section) にだけ番号をつける
secnolevel: 3    ←項 (Subsection) にも番号をつける
```

項を目次を含める

項 (Subsection) を目次に出すには、「config.yml」の設定項目「toclevel:」を「3」に変更します。

▼ ファイル「config.yml」

```
# 目次として抽出する見出しレベル
#toclevel: 2      ← (部と) 章と節までを目次に載せる
toclevel: 3      ← (部と) 章と節と項を目次に載せる
```

[PDF] 項への参照に節タイトルを含める

項 (Subsection) に番号がついていない場合、「@<secref>{...}」で項を参照すると表示が番号なしの項タイトルだけになるので、ユーザはその項を探せません。

たとえば前の項のタイトルは「項を目次を含める」です。もし

■ 詳しくは「4.5.7 項を目次を含める」を参照してください。

と書かれてあれば、項番号を手がかりにして読者はその項を簡単に探せるでしょう。しかし、もし項番号がなくて

■ 詳しくは「項を目次を含める」を参照してください。

と書かれてると、手がかりがないので読者はその項を探すのに苦労するでしょう。

このように、本文の中で項を参照しているなら項に番号をつけたほうがいいです。

Starter では別の解決策として、「@<secref>{...}」で項 (Subsection) を参照したときに親となる節 (Section) のタイトルもあわせて表示する機能を用意しています。たとえばこのように：

■ 詳しくは「4.5 見出し」の中の「項を目次を含める」を参照してください。

このような表示にするには、「config-starter.yml」の設定項目「secref_parenttitle:」を「true」に設定してください。

```
## @<secref>{}において、親の節のタイトルを含めるか？
secref_parenttitle: true # trueなら親の節のタイトルを含める
```

なお以前は、この値はデフォルトで「true」になっていました。しかし「@<secref>{}」での参照にページ番号がつくこと、また項タイトルにクローバーをつければ項であることがすぐ分かることから、現在ではデフォルトで「false」になっています。

項タイトルの記号を外す

項 (Subsection) のタイトル先頭につく記号 (クローバー、クラブ) は、「config-starter.yml」でオン・オフできます。

▼ ファイル「config-starter.yml」

```
## 項 (Subsection) タイトルの装飾
## (none: なし、symbol: 先頭にクローバー)
subsection_decoration: symbol
```

[PDF] 項タイトルの記号をクローバーから変更する

項 (Subsection) のタイトル先頭につく記号は、「sty/starter-heading.sty」で次のように定義されています。

▼ ファイル「sty/starter-heading.sty」

```
\newcommand{\starter@subsection@symbol}{\clubsuit}% クローバー
```

これを変更するには、「sty/mystyle.sty」で上書きしましょう。

▼ ファイル「sty/mystyle.sty」

```
% 「\newcommand」ではなく「\renewcommand」を使うことに注意
\renewcommand{\starter@subsection@symbol}{\heartsuit}% ハート
```

4.6 本

本のタイトルや著者名を変更する

本のタイトルや著者名は、「config.yml」で設定できます。

▼ ファイル「config.yml」

```
# 書名
booktitle: |-
  Re:VIEW Starter
  ユーザーズガイド
subtitle: |-
  便利な機能を一挙解説！

# 著者名。「 , 」で区切って複数指定できる
aut:
  - name: カウプラン機関極東支部
    file-as: カウプランキカンキョクトウシブ
```

Starter では、本のタイトルやサブタイトルを複数行で設定できます。こうすると、大扉（タイトルページ）でも複数行で表示されます。

[PDF] 表紙や大扉や奥付となる PDF ファイルを指定する

Starter では、本の表紙や大扉（タイトルページ）や奥付となる PDF ファイルを指定できます。config-starter.yml で以下の項目を変更してください*8。

▼ ファイル「config-starter.yml」

frontcover_pdffile: cover_a5.pdf	←表紙
backcover_pdffile: null	←裏表紙
titlepage_pdffile: null	←大扉
colophon_pdffile: null	←奥付
#includepdf_option: "offset=-2.3mm 2.3mm"	←位置を微調整する場合
includepdf_option: null	←通常は無指定でよい

いくつか注意点があります。

- PDF ファイルは、「images/」フォルダ直下に置いてください。
- 画像ファイルは使用できません。必ず PDF ファイルを使ってください。
- 1 つの PDF ファイルにつき 1 ページだけにしてください。もし PDF ファイルが複数ペー

*8 これらの設定は、以前は config.yml で行っていましたが、現在は config-starter.yml で行うように変更されました。また config.yml に古い設定が残っているとコンパイルエラーになり、「古い設定が残っているので移行するように」というエラーメッセージが出ます。

ジを持つ場合は、「cover.pdf<2>」のように取り込むページ番号を指定してください（実験的機能）。

- 複数の PDF ファイルを指定するには、YAML の配列を使って「[file1.pdf, file2.pdf]」のように指定します。「,」のあとには必ず半角空白を入れてください。
- もし位置がずれる場合は、「coverpdf_option:」の値を調整してください（以前は微調整が必要でしたが、現在は必要ないはずです）。

なお表紙と裏表紙がつくのは電子用 PDF ファイルの場合のみであり、印刷用 PDF ファイルにはつかない（し、つけてはいけない）ので注意してください。詳しくは「2.5 印刷用 PDF と電子用 PDF」（p.33）を参照してください。これに対して大扉と奥付は、指定されれば印刷用 PDF でも電子用 PDF でもつきます。

..... PNG や JPG の画像を PDF に変換する

macOS にて PNG や JPG を PDF にするには、画像をプレビュー.app で開き、メニューから「ファイル > 書き出す... > フォーマット:PDF」を選んでください。

macOS 以外の場合は、「画像を PDF に変換」などで Google 検索すると変換サービスが見つかります。

[PDF] 表紙の PDF から塗り足しを除いて取り込む

表紙や裏表紙の PDF ファイルを印刷所に入稿するには、通常の A5 や B5 に上下左右 3mm ずつを加えたサイズの PDF ファイルを使います。この上下左右 3mm の部分を「塗り足し」といいます*9。

つまり印刷用の表紙や裏表紙は、塗り足しの分だけ縦横サイズが少し大きいのです。そのため印刷用の表紙や裏表紙を電子用 PDF に取り込むには、塗り足し部分を除いたうえで取り込む必要があります。

Starter では PDF ファイル名の後ろに「*」をつけると、塗り足し部分を除いて取り込んでくれます。PDF のページ番号も一緒に指定する場合は「cover.pdf<2>*」のように指定してください。

▼ ファイル「config-starter.yml」

frontcover_pdffile: frontcover_a5.pdf*	←塗り足し部分を取り除く
backcover_pdffile: backcover_a5.pdf*	←塗り足し部分を取り除く
titlepage_pdffile: titlepage_a5.pdf	←大扉は塗り足しなし
colophon_pdffile: colophon_a5.pdf	←奥付は塗り足しなし

より正確には次のような動作をします。

*9 塗り足しについては「[PDF] 塗り足しをつける」（p.142）も参照してください。

- PDF ファイル名の最後に「*」がついてなければ、用紙サイズに合わせて PDF を拡大・縮小して、中央に配置します。
- PDF ファイル名の最後に「*」がついていると、拡大・縮小はせずにそのまゝの大きさで中央に配置します。その結果、塗り足し部分だけが用紙からはみ出るので、あたかも塗り足し部分が取り除かれたように見えます。

塗り足しつきで表紙を取り込むと上下に空きが入る理由

ファイル名の最後に「*」をつけない場合、用紙サイズに合わせて PDF を拡大または縮小しますが、縦横の比率は変えません。そのため、PDF が用紙より縦長だと上下を揃えようとして左右に空きが入り、PDF が用紙より横長だと左右を揃えようとして上下に空きが入ります。塗り足しのついた PDF ファイルをそのまま取り込むと上下が少し空くのは、PDF の縦横サイズが塗り足しの分だけ用紙より横長になるからです。

[PDF] 大扉のデザインを変更する

大扉（タイトルページ）の PDF ファイルを指定しなかった場合は、デフォルトデザインの大扉がつきます。このデザインは「sty/mytitlepage.sty」で定義されているので、デザインを変更したい場合はこのファイルを編集してください。編集する前にバックアップを取っておくといいいでしょう^{*10}。

```
$ cp sty/mytitlepage.sty sty/mytitlepage.sty.original
$ vi sty/mytitlepage.sty
```

[PDF] 奥付のデザインを変更する

奥付の PDF ファイルを指定しなかった場合は、デフォルトデザインの奥付がつきます。このデザインは「sty/mycolophon.sty」で定義されているので、デザインを変更したい場合はこのファイルを編集してください。編集する前にバックアップを取っておくといいいでしょう^{*11}。

```
$ cp sty/mycolophon.sty sty/mycolophon.sty.original
$ vi sty/mycolophon.sty
```

[PDF] 奥付を改ページしない

Starter では、奥付は自動的に最終ページに配置されます（詳しくは「奥付は最後のページに置く」(p.211) を参照してください）。この自動での配置を止めて手動でコントロールしたい場合は、次のどちらかを行ってください。

^{*10} Gitなどでバージョン管理している場合は、バックアップする必要はありません。

^{*11} Gitなどでバージョン管理している場合は、バックアップする必要はありません。

- 「sty/mycolopho.sty」において、7行目付近にある「\reviewcolophon」の先頭に「%」をつけて「%\reviewcolophon」とする。
- 「sty/mystyle.sty」に、「\def\reviewcolophon{ }」という記述を追加する。

たとえば、最終ページの上半分に著者紹介と過去の著作一覧を表示し、下半分に奥付を表示したいとします。その場合は奥付が自動的に改ページされないよう変更してから、あとがきを次のようにするといいでしょう。

▼ ファイル「contents/99-postface.re」：あとがき

```
= あとがき
....文章....

//clearpage      ←改ページ
== 著者紹介
....紹介文....

* 『著作1』
* 『著作2』
```

[PDF] 奥付に発行者と連絡先を記載する

「技書博^{*12}」という技術系同人誌イベントでは、奥付に「発行者」と「連絡先」の記載が必要となりました^{*13}。Starter では、奥付に発行者と連絡先を載せる設定を用意しています。

▼ ファイル「config.yml」

```
additional:
- key:      発行者
  value:     xxxxxx      ←サークル名ではなく個人名（ペンネーム可）
- key:      連絡先
  value:
    - xxxxx@example.com      ←メールアドレス（推奨）
    - https://www.example.com/ ← WebサイトURL
    - "@xxxxx"               ← Twitterアカウント（頭に '@' が必要）
```

連絡先にはいくつか補足事項があります。

- 連絡先には複数の値が指定できます。技書博によればメールアドレスがお勧めだそうです。
- 連絡先に URL が指定されると、自動的にリンクになります。
- 連絡先に Twitter アカウントを指定するには、先頭に「@」をつけてください。また Twitter アカウントも自動的にリンクになります。

^{*12} <https://gishohaku.dev>

^{*13} <https://esa-pages.io/p/sharing/13039/posts/115/7fbe936149a836eac678.html#%E5%A5%A5%E4%BB%98%E3%81%AE%E8%A8%98%E8%BC%89>

- 発行者や連絡先は、奥付の中では印刷所名の直前に置かれます。
- PDF の奥付にのみ表示されます。HTML や ePub へは未対応です。

なおこの設定には、発行者や連絡先以外の項目も追加できます。もし奥付に表示したい項目があれば、ここに追加してください。

[PDF] 注意事項の文章を変更する

大扉（タイトルページ）の直後のページには、免責や商標に関する以下のような注意事項が書かれています。

■免責

本書は情報の提供のみを目的としています。

本書の内容を実行・適用・運用したことで何が起きようとも、それは実行・適用・運用した人自身の責任であり、著者や関係者はいかなる責任も負いません。

■商標

本書に登場するシステム名や製品名は、関係各社の商標または登録商標です。

また本書では、TM、®、© などのマークは省略しています。

この文言を変更したい場合は、「sty/mytitlepage.sty」を編集してください。

[PDF] 注意事項の文章を出さない

大扉（タイトルページ）の裏ページにある注意事項の文章を出さないようにするには、次のどちらかを行ってください。

- 「sty/mytitlepage.sty」に お い て、「\newcommand{\mytitlenextpage}」を「\newcommand{\@mytitlenextpage}」に変更する。
- 「sty/mystyle.sty」において、「\let\mytitlenextpage=\undefined}」という記述を追加する。

4.7 索引

[PDF] 用語のグループ化を文字単位にする

索引ページでは、用語が「あ か さ た な は ……」のように行単位でグループ化されます。これを「あ い う え お か き ……」のように文字単位でグループ化するには、`config.yml` の「`makeindex_options: "-q -g"`」から「`-g`」を取り除いてください。

[PDF] 連続したピリオドのかわりに空白を使う

索引ページにて用語とページ番号との間は、デフォルトでは連続したピリオドで埋めています。これを空白で埋めるよう変更するには、`sty/indexsty.ist` を編集してください。

▼ `sty/indexsty.ist`

```
## 用語とページ番号との間をピリオドで埋める
#delim_0 "\\quad\\dotfill ~"
#delim_1 "\\quad\\dotfill ~"
#delim_2 "\\quad\\dotfill ~"

## 用語とページ番号との間を空白で埋める
delim_0 "\\quad\\hfill"
delim_1 "\\quad\\hfill"
delim_2 "\\quad\\hfill"
```

なお `sty/indexsty.ist` の中の設定項目については、`mendex` コマンドのマニュアル^{*14}を参照してください。`mendex` コマンド（または `upmendex` コマンド）のオプションについても記載されています。

[PDF] 用語見出しのデザインを変更する

索引ページの用語見出しは「■あ」「■か」のようになっています。このデザインは `sty/starter-misc.sty` の「`\starterindexgroup`」で設定されています。変更するには「`\starterindexgroup`」の定義を `sty/starter-misc.sty` から `sty/mystyle.sty` にコピーして、「`\newcommand`」を「`\renewcommand`」に変更して中身をカスタマイズしてください。

▼ `sty/mystyle.sty`

```
\renewcommand{\starterindexgroup}[1]{% 索引グループ
{%
  \bfseries%                % 太字
  \gtfamily\sffamily%        % ゴシック体
  ■#1%                       % 例：`■あ`
```

^{*14} <http://tug.ctan.org/info/mendex-doc/mendex.pdf>

```
}%
}
```

また `sty/indexsty.ist` を編集すると、別の \LaTeX マクロ名を指定できます。

[PDF] その他

- 索引ページにおいて子要素の用語で使う「——」は、`sty/starter-misc.sty` の「`\starterindexplaceholder`」で定義されています。
- 「`@<term>{ }`」で表示するフォントは、`sty/starter-misc.sty` の「`\starterterm`」で定義されています。デフォルトではゴシック体にて表示するように定義されています。
- 用語の転送先を表す「→」を別の記号や文字列に変更するには、`sty/mystyle.sty` にたとえば「`\renewcommand{\seename}{\textit{see: } }`」のように書きます。

4.8 Rake タスク

デフォルトタスクを設定する

Starter では、rake コマンドのデフォルトタスクを環境変数「\$RAKE_DEFAULT」で指定できます。

▼ rake のデフォルトタスクを設定する

```
$ rake          ←デフォルトではタスク一覧が表示される
$ export RAKE_DEFAULT=pdf ←環境変数を設定
$ rake          ←「rake pdf」が実行される
```

独自のタスクを追加する

Starter では、ユーザ独自の Rake タスクを追加するためのファイル「lib/tasks/mytasks.rake」を用意しています。

▼ ファイル「lib/tasks/mytasks.rake」

```
## 独自のタスクを追加する
desc "ZIPファイルを生成する"
task :zip do
  rm_rf "mybook"
  mkdir "mybook"
  cp_r ["README.md", "mybook.pdf"], "mybook"
  sh "zip -r9 mybook.zip mybook"
end
```

コンパイルの前処理を追加する

Starter では、任意の処理をコンパイルの前に実行できます。

▼ ファイル「lib/tasks/mytasks.rake」

```
def my_preparation(config)      # 新しい前処理用関数
  print "...前処理を実行...\n"
end

PREPARES.unshift :my_preparation # 前処理の先頭に追加
```

これを使うと、たとえば以下のようなことができます。

- 原稿ファイルをコピーし書き換える。
- 複数の原稿ファイルを結合して1つにする。
- 1つの原稿ファイルを複数に分割する。

第 5 章

FAQ

よくある質問とその回答 (Frequently Asked Question, FAQ) を紹介します。

5.1 コンパイルエラー

コンパイルエラーが出たとき

PDF ファイル生成時のコンパイルエラーについて、対処方法を説明します。

PDF へのコンパイルは、内部では大きく 3 つのステージに分かれます。

- (1) 設定ファイルを読み込む (`config.yml`、`config-starter.yml`、`catalog.yml`)
- (2) 原稿ファイル (`*.re`) を \LaTeX ファイルへ変換する
- (3) \LaTeX ファイルから PDF ファイルを生成する

(1) のステージでは、設定ファイルの書き方に間違いがあるとエラーが発生します。たとえば次のような点に注意してください。

- タブ文字を使ってないか (使っているとエラー)
- インデントが揃っているか (揃ってないとエラー)
- 必要な空白が抜けてないか (「-」や「:」の直後)

(2) のステージでは、インライン命令やブロック命令の書き方に間違いがあるとエラーが発生します。たとえば次のような点に注意してください。

- インライン命令がきちんと閉じているか
- ブロック命令の引数が足りているか、多すぎないか
- 「`//}`」が足りてないか、または多すぎないか
- 「`@<fn>{}`」や「`@{}`」や「`@<table>{}`」のラベルが合っているか
- 「`@<chapref>{}`」で指定した章 ID が合っているか
- 「`@<secref>{}`」で指定した節や項が存在するか
- 脚注の中で「`]`」を「`\]`」とエスケープしているか
- 「`//image`」で指定した画像ファイルが存在するか

- 原稿ファイル名を間違っていないか
- 原稿ファイルの文字コードが UTF-8 になっているか

このエラーの場合はエラーメッセージが出るので、それを注意深く読めば解決の糸口が掴めます。

(3) のステージでのエラーはあまり発生しないものの、原因がとても分かりにくく、 \LaTeX に慣れている人でないと解決は難しいです。ときどきキャッシュファイル (*.pdf) のせいでおかしくなることがあるので、**困ったらキャッシュファイルを消して再コンパイル**してみましょう。

▼ 困ったらキャッシュファイルを消して再コンパイル

```
$ rm -rf mybook-pdf ←キャッシュファイルを消す
$ rake pdf           ←または rake docker:pdf
```

それでもエラーが解決できない場合は、ハッシュタグ「#reviewstarter」をつけて Twitter で聞いてみてください（宛先はなくて構いません）。

「review-pdfmaker」で PDF が生成できない

Starter では、「review-pdfmaker」や「review-epubmaker」は使えません。かわりに「rake pdf」や「rake epub」を使ってください。

5.2 本文

左右の余白が違う

印刷用 PDF ファイルでは、左右の余白幅を意図的に変えています。これは印刷時の読みやすさを確保したまま、本文の幅を最大限に広げているからです。詳しくは「[PDF] 印刷用では左右の余白幅を充分にとる」(p.206) をご覧ください。

文章中のプログラムコードに背景色をつけたい

「`@<code>{...}`」による文章中のプログラムコードに、背景色をつけられます。そのためには、「config-starter.yml」で「`inlinecode_gray: true`」を設定してください。

▼ ファイル「config-starter.yml」

```
inlinecode_gray: true
```

こうすると、sty/starter.sty において次のような L^AT_EX マクロが有効になります。背景色を変えたい場合はこのマクロを変更してください。

▼ ファイル「sty/starter.sty」

```
\renewcommand{\reviewcode}[1]{%
  \,%                               % ほんの少しスペースを入れる
  \colorbox{shadecolor}{%          % 背景色を薄いグレーに変更
    \texttt{#1}%                   % 文字種を等幅フォントに変更
  }%
  \,%                               % ほんの少しスペースを入れる
}
```

索引をつけたい

Re:VIEW の Wiki ページを参照してください。

- <https://github.com/kmuto/review/blob/master/doc/makeindex.ja.md>

5.3 プログラムコードとターミナル

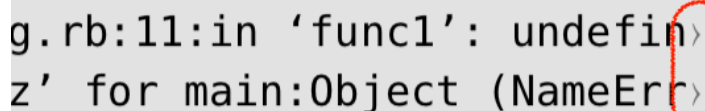
プログラムコードの見た目が崩れる

おそらく、プログラムコードの中でタブ文字が使われています。タブ文字を半角空白に置き換えてみてください。

Starter では、プログラム中のタブ文字は自動的に半角空白に置き換わります。しかしインライン命令があるどうしてもカラム幅を正しく計算できないので、意図したようには半角空白に置き換わず、プログラムの見た目が崩れます。

右端に到達してないのに折り返しされる

Starter ではプログラム中の長い行が自動的に右端で折り返されます。このとき、右端にはまだ文字が入るだけのスペースがありそうなのに折り返しされることがあります (図 5.1)。



▲ 図 5.1: 右端にはまだ文字が入るだけのスペースがありそうだが…

このような場合、プログラムやターミナルの表示幅をほんの少し広げるだけで、右端まで文字が埋まるようになります。

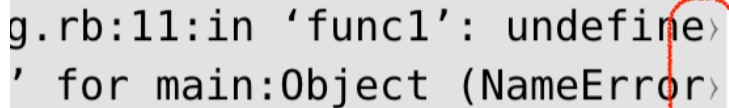
具体的には、ファイル「config-starter.yml」の中の「program_widen: 0.0mm」や「terminal_widen: 0.0mm」を、たとえば「0.3mm」に設定してください (値は各自で調整してください)。

▼ ファイル「config-starter.yml」

```
## プログラム (//list) の表示幅をほんの少しだけ広げる。
program_widen: 0.0mm
program_widen: 0.3mm

## ターミナル (//terminal, //cmd) の表示幅をほんの少しだけ広げる。
terminal_widen: 0.0mm
terminal_widen: 0.3mm
```

こうすると、プログラムやターミナルの表示幅が少しだけ広がり、文字が右端まで埋まるようになります (図 5.2)。



```
g.rb:11:in 'func1': undefine>
' for main:Object (NameError>
```

▲ 図 5.2: 表示幅をほんの少し広げると、右端まで埋まるようになった

全角文字の幅を半角文字 2 個分にしたい

「`//list`」や「`//terminal`」の第 3 引数に「`widecharfit=on`」を指定すると、全角文字の幅が半角文字 2 文字分になります。すべての「`//list`」や「`//terminal`」でこのオプションを有効化したい場合は、`config-starter.yml` の「`program_default_options:`」や「`terminal_default_options:`」に、「`widecharfit: on`」を指定してください。

または英数字の幅が狭いフォントを使うと、半角文字の幅が全角文字の約半分になります（ぴったりにはなりません）。`config-starter.yml` で、次のうち必要なほうを設定してください。

```
## //list用
program_ttfont: inconsolata-narrow

## //terminal用
terminal_ttfont: inconsolata-narrow
```

ターミナルの出力を折り返しせずに表示したい

ターミナルの出力結果を折り返しせずに表示するには、いくつか方法があります。

小さいフォントで表示する

「`//terminal`」や「`//list`」では、小さいフォントで表示するオプションがあります。フォントサイズは「`small`」「`x-small`」「`xx-small`」の 3 つが選べます。

▼ サンプル

```
//terminal[][小さいフォントで表示する][fontsize=x-small]{
    Table "public.customers"
      Column | Type | Nullable | Default
      -----+-----+-----+-----
      id      | integer | not null | nextval('customers_id_seq'::regclass)
      name    | text   | not null |
      created_on | date   | not null | CURRENT_DATE
Indexes:
```

```
"customers_pkey" PRIMARY KEY, btree (id)
"customers_name_key" UNIQUE CONSTRAINT, btree (name)
//}
```

▼ 表示結果

▼ 小さいフォントで表示する

```
Table "public.customers"
  Column | Type   | Nullable | Default
-----+-----+-----+-----
id       | integer | not null | nextval('customers_id_seq'::regclass)
name     | text    | not null |
created_on | date   | not null | CURRENT_DATE
Indexes:
    "customers_pkey" PRIMARY KEY, btree (id)
    "customers_name_key" UNIQUE CONSTRAINT, btree (name)
```

幅の狭い等幅フォントを使う

Starter では等幅フォントを複数の中から選べます。このうち「inconsolata-narrow」は文字の表示幅が狭いので、できるだけ折り返しをさせたくない場合に有効です。

「config-starter.yml」で「terminal_ttfont:」を「inconsolata-narrow」に設定してください。

コードハイライトしたい

コードハイライトは、Starter では未サポートです*1。将来的にはサポートする予定ですが、時期は未定です。

*1 Re:VIEW ではコードハイライトができるそうです。

5.4 画像

PDF と HTML とで画像の表示サイズを変えたい

Re:VIEW や Starter では、PDF と HTML とで画像の表示サイズを別々に指定できます。たとえば次のサンプルでは、PDF (L^AT_EX) では本文幅いっぱい、HTML では本文幅の半分の大ききで画像を表示します。

▼ サンプル

```
//image[dummy-image][表示幅を変える][latex::scale=1.0,html::scale=0.5]
```

▼ 表示結果



▲ 図 5.3: 表示幅を変える

なお「`latex::xxx`」や「`html::xxx`」のような指定方法は、他のオプションやコマンドでも使えます。

印刷用 PDF と電子用 PDF とで異なる解像度の画像を使いたい

印刷用 PDF では解像度の高い画像を使いたい、けど電子用 PDF では画像の解像度を低くしてデータサイズを小さくしたい、という場合は、「`images/`」フォルダごと切り替えるのがいいでしょう。具体的には次のようにします。

▼ 高解像度と低解像度の画像を切り替える (macOS または Linux の場合)

```
### 事前準備
$ mkdir images_highres ←解像度の高い画像用のフォルダを作る(印刷用)
$ mkdir images_lowhres ←解像度の低い画像用のフォルダを作る(電子用)
$ mv images images.bkup ←既存のimagesフォルダを削除するか別名にする

### 解像度の高い画像に切り替えて印刷用PDFを生成
$ rm -f images ←シンボリックリンクを消す
$ ln -s images_highres images ←シンボリックリンクを作る
```

```

$ rake pdf t=pbook          ←印刷用PDFを生成する

### 解像度の低い画像に切り替えて電子用PDFを作成
$ rm -f images              ←シンボリックリンクを消す
$ ln -s images_lowres images ←シンボリックリンクを作る
$ rake pdf t=ebook          ←電子用PDFを生成する

```

切り替え作業が面倒なら、独自の Rake タスクを作成するといいいでしょう。「lib/tasks/mytasks.rake」に次のようなタスクを追加してください。

▼ lib/tasks/mytasks.rake

```

desc "印刷用PDFを生成"
task :pbook do
  rm_f 'images'
  ln_s 'images_highres', 'images'
  sh "rake pdf t=pbook"
  #sh "rake docker:pdf t=pbook"
end

desc "電子用PDFを生成"
task :ebook do
  rm_f 'images'
  ln_s 'images_lowres', 'images'
  sh "rake pdf t=ebook"
  #sh "rake docker:pdf t=ebook"
end

```

これにより、「rake pbook」で高解像度の画像を使って印刷用 PDF が生成され、また「rake ebook」で低解像度の画像を使って電子用 PDF が生成されます。

画像の解像度はどのくらいにすればいいか

画像の「解像度」(Resolution) とは、簡単にいえば画像の「きめ細かさ」です。解像度が高い(=きめ細かい)ほど画像のドットやギザギザがなくなり、解像度が低いほど画像のドットやギザギザが目立ちます。

解像度は「dpi」(dot per inch) という単位で表します。この値が大きいほどきめ細かく、低いほど粗くなります。望ましい解像度は用途によって異なります。大雑把には次のように覚えておくといいでしょう。

- 安い Windows ノート PC で表示するなら、72dpi で十分
- Macbook の Retina ディスプレイできれいに表示するなら、144dpi が必要
- 高級スマートフォンできれいに表示するなら、216dpi 必要
- 商業印刷で使うなら、カラー画像で 350dpi、モノクロ画像で 600dpi 必要

たとえば安い Windows ノート PC でスクリーンショットを撮ると、画像の解像度が 72dpi になります。これを Macbook やスマートフォンで等倍表示すると、必要な解像度に足りないせいで画像のドットが目立ちます*2。印刷すれば粗さがもっと目立ちます。

では、技術同人誌では画像の解像度をどのくらいにすればいいでしょうか。すでに説明したように、商業印刷に使う画像は 350dpi 以上の解像度が必要だとよく言われます。しかし（マンガやイラストの同人誌ではなく）技術系同人誌であれば、そこまで高解像度の画像を用意しなくても大丈夫です。

実際には、Retina ディスプレイや最近のスマートフォンで画像を等倍表示して気にならない解像度であれば、印刷してもあまり気になりません。つまり画像の解像度が 144dpi であれば、印刷しても問題は少ないし、電子用 PDF で使っても 350dpi と比べてデータサイズをかなり小さくできます。

印刷用と電子用とで解像度の違う画像を用意するのは面倒なので、画像の解像度を 144dpi にして印刷用と電子用とで同じ画像を使うのが簡単かつ妥当な方法でしょう。

白黒印刷用の PDF にカラー画像を使っても大丈夫か

白黒印刷用の PDF にカラー画像を使っても、たいていは大丈夫です。たとえばカラーのスクリーンショット画像を、わざわざモノクロ画像に変換する必要はありません。

ただし、白黒印刷すれば当然ですがカラーではなくなるので、**色の違いだけで見分けをつけていたものは白黒印刷すると見分けがつかなくなります**。形を変えたり線の種類を変えるなどして、白黒になっても見分けがつくような工夫をしましょう。

また図の一部を赤い丸や四角で囲んでいる場合、白黒印刷すると赤ではなくなるので、たとえば「赤い丸で囲った部分に注目してください」という文章が読者に通じなくなります。これもよくある失敗なので気をつけましょう。

*2 等倍で表示すると荒い画像でも、大きい画像を縮小して表示すれば粗さは目立たなくなります。解像度の高い画像を用意できない場合は、かわりに大きい画像を用意して印刷時に縮小するといいでしょ。

5.5 ノンブル

ノンブルとは

「ノンブル」とは、すべてのページにつけられた通し番号のことです。ページ番号と似ていますが、次のような違いがあります。

通し番号

- ページ番号は、まえがきや目次では「i, ii, iii, ...」で、本文が始まると「1, 2, 3, ...」とまる。つまり通し番号になっていない。
- ノンブルは、まえがきや目次や本文に関係なく「1, 2, 3, ...」と続く通し番号になっている。

用途

- ページ番号は、読者が目的のページへ素早くアクセスするために必要。
- ノンブルは、印刷所がページの印刷順序を間違わないために必要。

表示位置

- ページ番号は読者のために存在するので、読者からよく見える位置に置く。
- ノンブルは印刷所の人だけが見られればよく、読者からは見えないほうがよい。

ノンブルは、印刷所へ入稿するときに必要です。印刷しないならノンブルは不要であり、電子用 PDF にはノンブルはつけません。

ノンブルについてもっと知りたい場合は「ノンブル 同人誌」で Google 検索すると詳しい情報が見つかります。

ノンブルを必要とする印刷所

たいていの印刷所では入稿する PDF にノンブルが必要ですが、必要としない印刷所もあります。たとえば技術書典でよく利用される印刷所のうち、「日光企画」ではノンブルが必須で、「ねこのしっぽ」では不要（だけどあると望ましい）です。主な同人誌向け印刷所の情報を表 5.1 にまとめたので参考にしてください。

このように印刷所によって違いますが、**ノンブル不要の印刷所であってもノンブルをつけるほうが望ましい**です。強い理由がないなら、印刷用 PDF にはノンブルをつけましょう。

ノンブルのつけ方

Re:VIEW と Starter のどちらも、印刷用 PDF にはノンブルが自動的につきます（電子用にはつきません）。Starter では、印刷用 PDF ならページの右下隅または左下隅に、グレーの通し番号がついているはずです。これがノンブルです。

ノンブルは印刷所の人だけが見られればいいので、読者からは見えにくい位置（見開きの内側）に置かれます。このように読者から見えにくい位置に置かれたノンブルを、特に「隠しノンブル」といいます。

▼ 表 5.1: 同人誌向け印刷所ごとのノンブル必要・不要情報

印刷所	ノンブル	参考 URL
日光企画	必要	http://www.nikko-pc.com/qa/yokuaru-shitsumon.html#3-1
ねこのしっぽ	不要	https://www.shippo.co.jp/neko/faq_3.shtml#faq_039
しまや出版	必要	https://www.shimaya.net/howto/data-genkou.html#no
栄光	必要	https://www.eikou.com/qa/answer/66
サンライズ	必要	https://www.sunrisep.co.jp/09_genkou/002genko_kiso.html#title-6
オレンジ工房	必要	https://www.orangekoubou.com/order/question.php#article05
太陽出版	必要	https://www.taiyoushuppan.co.jp/doujin/howto/nombre.php
ポプルス	不要	https://www2.popls.co.jp/pop/genkou/q_and_a.html#nombre
丸正インキ	必要	https://www.marusho-ink.co.jp/howto/nombre.html
トム出版	必要	https://www.tomshuppan.co.jp/manual/data.html
金沢印刷	必要	http://www.kanazawa-p.co.jp/howtodata/howtodata_kihon-rule.html
ケーナイン	必要	https://www.k-k9.jp/data/nombre/
booknext	必要	https://booknext.ink/guide/making/

また Starter では、すでに生成済みの PDF に対してあとからノンブルを追加できます。たとえば大扉（タイトルページ）や奥付を Illustrator や Keynote で作成し^{*3}、それを使って印刷用 PDF の大扉や奥付のページを手動で入れ替えるような場合は^{*4}、入れ替えたページにノンブルがつきません。このような場合は、次のような作業手順になります。

- (1) 印刷用 PDF をノンブルなしで生成する。
- (2) 手動で大扉や奥付のページを入れ替える。
- (3) 生成済みの印刷用 PDF にノンブルをつける。

ここで (1) 印刷用 PDF をノンブルなしで生成するには、`config-starter.yml` で「`nombre: off`」を設定します。また (3) 生成済みの印刷用 PDF にノンブルをつけるには、ターミナル画面で「`rake pdf:nombre`」を実行してください。

▼ 生成済みの印刷用 PDF にノンブルをつける

```
$ rake pdf:nombre          ← PDFの全ページにノンブルをつける
$ rake docker:pdf:nombre   ← Dockerを使っている場合はこちら
```

この Rake タスクでは、ノンブルをつけたい PDF ファイル名と出力先の PDF ファイル名を指定できます。たとえば `mybook.pdf` にノンブルをつけたものを `mybook_nombre.pdf` として出力するには、次のようにします。

^{*3} 大扉（タイトルページ）を別のソフトで作成した例が「大扉を別のソフトで作成する」（p.209）にあります。

^{*4} 昔はこうする必要がありましたが、現在の Starter は大扉と奥付を PDF ファイルで用意すればそれを読み込めるので、手動で入れ替える必要はなくなりました。詳しくは「[PDF] 表紙や大扉や奥付となる PDF ファイルを指定する」（p.157）を参照してください。

▼ 生成済みの印刷用 PDF にノンブルをつける

```
$ rake pdf:nombre file=mybook.pdf out=mybook_nombre.pdf
```

「out=...」が指定されなければ、「file=...」と同じファイル名が使われます。また「file=...」が指定されなければ、「rake pdf」で生成される PDF ファイルの名前が使われます。

なお「rake pdf:nombre」はノンブルをつけるだけであり、再コンパイルはしないので注意してください。また「PDFOperation^{*5}」を使っても、PDF ファイルにノンブルが簡単につけられます。

.....

ノンブル用フォントの埋め込み

残念ながら、「rake pdf:nombre」でノンブルをつけるとそのフォントが PDF ファイルに埋め込まれません（これは PDF ファイルを操作しているライブラリの限界によるものです）。そのため、印刷所へ入稿するまえにフォントを埋め込む必要があります。

対策方法は <https://kauplan.org/pdfoperation/> を見てください。また印刷用 PDF にデフォルトでつくノンブルならこのような問題はありません。

.....

ノンブルの調整

印刷所によっては、ノンブルの位置や大きさを指定される場合があります。その場合は config-starter.yml の中の、ノンブルに関する設定項目を調整してください。なおこの設定は、「rake pdf:nombre」タスクでも使われます。

▼ config-starter.yml：ノンブルに関する設定項目

```
## ノンブル（1枚目からの通し番号）の設定
nombre: on          ← on ならノンブルをつける
nombre_sidemargin: 0.5mm ←ページ左右からのマージン幅
nombre_bottommargin: 2.0mm ←ページ下からのマージン高
nombre_fontcolor: gray ←'gray' or 'black'
nombre_fontsize: 6pt  ←フォントサイズ（6～8pt）
nombre_startnumber: 1  ←ノンブルの開始番号（3から始める流儀もある）
```

ノンブルの注意点

印刷所への入稿に慣れていないと、ノンブルに関するトラブルを起こしがちです。初心者の方は次のような点に注意してください。

- すでに説明したように、ノンブルは印刷するときが必要であり、印刷しなければ不要です。Starter では印刷用 PDF にのみ自動でノンブルがつきますが、電子用 PDF にはつきません。

^{*5} <https://kauplan.org/pdfoperation/>

- ノンブルは、表紙と裏表紙には必要ありません。印刷所へ入稿するとき、表紙と裏表紙の入稿は本文とは別に行います。そのため、表紙と裏表紙にはノンブルをつける必要はありませんし、つけてはいけません。
- ノンブルは、表紙と裏表紙を除いたすべてのページに必要です。たとえ空白ページであってもノンブルを省略してはいけません。
- 別のソフトで作った大扉（タイトルページ）や奥付を PDF に入れた場合は、それらのページにもノンブルをつけることを忘れないでください。
- 印刷所からノンブルの位置や大きさの指定がある場合は、それに従ってください。ノンブルの調整方法は `config-starter.yml` で行えます。
- ノンブルは「1」から始めることがほとんどですが、印刷所によっては「3」から始めるよう指示されることがあります。その場合は `config-starter.yml` に「`nombre_startnumber: 3`」を設定してください。
- ノンブルのフォントが PDF に埋め込まれていることを確認してください。ただし本文のフォントほど重要ではないので、もしノンブルのフォントがうまく埋め込めない場合は、そのままで入稿できないか印刷所に相談してみましょう。

5.6 L^AT_EX 関連

LuaLaTeX と jreq を使いたい

Starter では、LuaLaTeX と jreq.cls にはまだ対応していません。どうしてもこれらを使いたい場合は、Re:VIEW が対応しているのでそちらを使うといいでしょう。

LuaLaTeX は好きなフォントが簡単に使えるという大きな利点がありますが、コンパイルが遅くなるという欠点があります。また jreq.cls はカスタマイズ方法がよく分かっていないので、採用ができません。

これらの採用は今後の課題です。

L^AT_EX のスタイルファイルから環境変数を参照する

Starter では、名前が「STARTER_」で始まる環境変数を L^AT_EX のスタイルファイルから参照できます。

たとえば「STARTER_FOO_BAR」という環境変数を設定すると、sty/mystyle.sty や sty/starter.sty では「\STARTER@FOO@BAR」という名前で参照できます。この例で分かるように、環境変数名の「_」は「@」に変換されます。

▼ 環境変数を設定する例 (macOS, UNIX)

```
$ export STARTER_FOO_BAR="foobar"
```

▼ 環境変数を参照する例

```
%% ファイル : sty/mystyle.sty
\newcommand\foobar[0]{%           % 引数なしコマンドを定義
  \@ifundefined{STARTER@FOO@BAR}{% % 未定義なら
    foobar%                       % デフォルト値を使う
  }{%                             % 定義済みなら
    \STARTER@FOO@BAR%             % その値を使う
  }%
}
```

この機能を使うと、出力や挙動を少し変更したい場合に環境変数でコントロールできます。また値の中に「\$」や「\」が入っていてもエスケープはしないので注意してください。

5.7 その他

高度なカスタマイズ

設定ファイルや L^AT_EX スタイルファイルでは対応できないようなカスタマイズが必要になることがあります。たとえば次のような場合です。

- 印刷用とタブレット用で PDF ファイルの仕様を大きく変えたい。
- B5 サイズと A5 サイズの両方の PDF ファイルを生成したい。
- β版と本番用とで設定を切り替えたい。
- コンパイルするたびに、あるプログラムの実行結果を原稿ファイルに埋め込みたい。

このような要件を、Re:VIEW や Starter の機能を使い倒して実現してもいいですが、それよりも汎用的なやり方で実現しましょう。方針としては、設定ファイルや原稿ファイルを用途に応じて生成するのがいいでしょう。

ここでは例として「印刷用とタブレット用で `config-starter.yml` の内容を変える」ことを考えてみます。

(1) 「`config-starter.yml`」に拡張子「`.eruby`」をつけます。

```
$ mv config-starter.yml config-starter.yml.eruby
## またはこうでもよい
$ mv config-starter.yml{,.eruby}
```

(2) 次に、そのファイルに次のような条件分岐を埋め込みます。

▼ ファイル「`config-starter.yml.eruby`」

```
....(省略)....
<% if buildmode == 'printing' # 印刷向け %>
  target:    pbook
  pagesize:  B5
  fontsize:  10pt
  textwidth: 44zw
<% elsif buildmode == 'tablet' # タブレット向け %>
  target:    tablet
  pagesize:  A5
  fontsize:  10pt
  textwidth: 42zw
<% else abort "error: buildmode=#{buildmode.inspect}" %>
<% end %>
```

(3) 「`config-starter.yml`」を生成する Rake タスクを定義します。ここまでが準備です。

▼ lib/tasks/mytasks.rake

```
def render_eruby_files(param)  # 要 Ruby >= 2.2
  Dir.glob('**/*.eruby').each do |erubyfile|
    origfile = erubyfile.sub(/\..eruby$/, '')
    sh "erb -T 2 -P '#{param}' #{erubyfile} > #{origfile}"
  end
end

namespace :setup do

  desc "*印刷用に設定 (B5, 10pt, mono)"
  task :printing do
    render_eruby_files('buildmode=printing')
  end

  desc "*タブレット用に設定 (A5, 10pt, color)"
  task :tablet do
    render_eruby_files('buildmode=tablet')
  end

end
```

(4) 「rake setup:printing」または「rake setup:tablet」を実行すると、config-starter.yml が生成されます。そのあとで「rake pdf」を実行すれば、用途に応じた PDF が生成されます。

```
$ rake setup:printing # 印刷用
$ rake pdf
$ mv mybook.pdf mybook_printing.pdf

$ rake setup:tablet   # タブレット用
$ rake pdf
$ mv mybook.pdf mybook_tablet.pdf
```

第 6 章

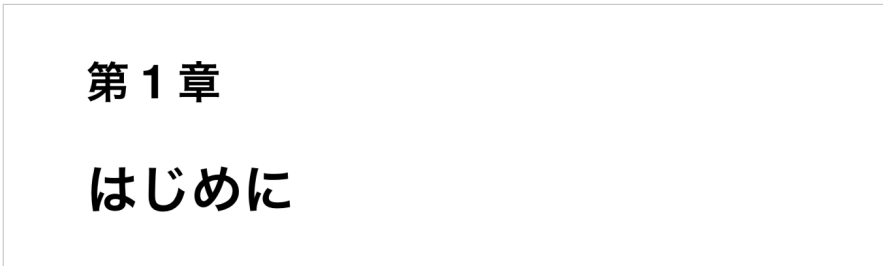
よりよい本づくり

せっかく本を作るなら、よりよい本づくりを目指しましょう。
この章では、よりよい本にするためのポイントを紹介します。

6.1 まえがき

まえがきには章番号をつけない

ときどき、「まえがき」に章番号がついている技術系同人誌を見かけます（図 6.1）。しかし一般的には、「まえがき」や「あとがき」には章番号はつけません。



第 1 章

はじめに

▲ 図 6.1: 間違って章番号がついた「まえがき」

Re:VIEW および Starter では、「catalog.yml」の「PREDEF:」や「POSTDEF:」に指定した原稿ファイルには、章番号がつきません。これを間違えて「CHAPS:」に含めてしまうと、「まえがき」や「あとがき」に章番号がついてしまうので気をつけてください。

まえがきに「本書の目的」を入れる

「まえがき」に本の概要や目的があると、本を手にとった人がその本を買うかどうかを判断できます。

次のは「本書の目的」のサンプルです。

▼ サンプル

本書の目的

本書の目的は、SQL を初めてさわる初心者が、簡単な検索と集計を SQL でできるようになることです。マーケティング部に配属された新卒 1 年目の人が、SQL で簡単なデータ分析をできるようになりたいなら、この本はぴったりです。

SQL のチューニングや、データベース設計といった内容は、本書の範囲ではありません。

まえがきに「対象読者」を入れる

先ほどの話と似ていますが、まえがきに「本の対象読者」を書きましょう。本を手にとった人が、自分がその本の対象読者かどうか判断できます。

このとき、「初心者」と「初級者」をちゃんと区別するといいでしょ。

- 「初心者」は入門書をまだ読んでいない人や読んでる途中の人
- 「初級者」は入門書を読み終えた人、実務での経験が浅い人

次のは「対象読者」のサンプルです。

▼ サンプル

本書の対象読者

本書は、何らかのオブジェクト指向言語の入門書を読み終えた初級者を対象としています。そのため、「オブジェクト」「クラス」「メソッド」などの用語は説明なしに使用します。まったくの初心者には本書の対象ではないので注意してください。

まえがきに「前提知識」を入れる

これまでの話と似ていますが、その本を読むのにどんな前提知識がどのくらいのレベルで必要かを「まえがき」に書いておくと、本を手にとった人は買うかどうかの判断がしやすいです。

このとき、具体例を添えておくといいでしょう。たとえば「Linux の基礎知識を前提とします」だけでは、どんな基礎知識がどのレベルで必要なのか、よくわかりません。もしこれが「`ln -s` でシンボリックリンクが作れること」「`ps -ef|grep gzip` が何をしているのか分かること」「パイプとリダイレクトの違いが説明できること」だと、必要な前提知識のレベルがよく分かります。

次のは「前提知識」のサンプルです。

▼ サンプル

本書の前提知識

本書は、Python をある程度使いこなしている中級者以上を対象にしています。そのため、Python の基礎知識を持っていることを前提とします。

具体的には、次のことがすべて分かることが前提となります。

- 「`[x for x in range(1, 11) if x % 2]`」の意味が分かる
- 「`(lambda x, y: x + y)(3, 4)`」の結果が予想できる
- 「`x=1`」と「`fn(x=1)`」の違いが分かる
- 「`return`」と「`yield`」の違いが説明できる
- 「`@property`」が何か分かる

まえがきにサポートサイトの URL を載せる

本の正誤表が載っていたり、サンプルコードがダウンロードできるサイトのことをここでは「サポートサイト」と呼ぶことにします。

本や同人誌を書いたら、ぜひサポートサイトを用意しましょう。そして正誤表とサンプルコード、できれば質問ができる連絡先を用意しましょう。

そしてその URL をまえがき（と奥付）に載せましょう。

次のはサポートサイト紹介文のサンプルです。

▼ サンプル

サポートサイト

本書の正誤表は次のサイトに載っています。サンプルコードもここからダウンロードできます。

- <https://www.example.com/mygreatbook/>

まえがきに謝辞を載せる

レビューしてくれた人や編集者がいたら、まえがきに謝辞を書きましょう。家族への謝辞を入れるのもよくあります。

謝辞を忘れるのはかなり失礼な行為に当たります。また名前を間違ったり、名前が抜けるのも大変失礼です。締切間際に謝辞を書く間違える可能性が高いので、執筆期間の早いうちに謝辞を書いておきましょう。

まえがきにソフトウェアのバージョンを載せる

まえがきに、使用した（または動作検証した）ソフトウェアのバージョンを載せるといいでしょう。

たとえば Python の 2.7 なのか 3.X なのかは大きな問題です。また Linux だと CentOS なのか Ubuntu なのか、Ubuntu なら 18.04 なのか 20.04 なのか、といった情報をまえがきに書いておきましょう。

まえがきの章タイトル以外は目次に載せない

目次には「まえがき」だけが載っていればよく、「本書の目的」や「対象読者」や「謝辞」などは目次に載せる必要はありません。

これらを目次に載せないための方法は2つあります。

ひとつは、節 (Section) や項 (Subsection) のタイトルに「[notoc]」オプションをつけることです。これをつけると、その節や項は目次に載りません。

```
= はじめに

==[notoc] 本書の目的

==[notoc] 対象読者

==[notoc] 謝辞
```

もうひとつは、節 (Section) や項 (Subsection) を飛ばして目 (Subsubsection) を使うことです。通常、目次に乗るのは項までで目は載りません。そこで、次のような見出しにすれば「本書の目的」や「対象読者」や「謝辞」は目次に載りません。

```
= はじめに

==== 本書の目的

==== 対象読者

==== 謝辞
```

この方法は見出しのレベルを飛ばしているので、本文で使うのはよくありません。しかし目次に載らないところで使うなら、あまり気にしなくてもいいでしょう。

6.2 初心者向け入門書

初心者向け入門書ではフォントを大きめにする

Starter では、デフォルトでは次のようなフォントサイズにしています。

- ページが B5 サイズなら、フォントは 10pt
- ページが A5 サイズなら、フォントは 9pt

しかし初心者向けの入門書では、次のように少し大きめのフォントを使ったほうが、紙面から受ける圧迫感が減ります*1。

- ページが B5 サイズなら、フォントは 11pt
- ページが A5 サイズなら、フォントは 10pt

とはいえフォントを大きくしていない入門書もよく見かけます。フォントを大きくするとページ数が増えてその分印刷代が高くなるので、どうするかはよく考えてください。

Starter で本文のフォントサイズを変える方法は、「[PDF] フォントサイズを変更する」(p.145)を参照してください。またそれに合わせて本文の幅も変更する必要があるので、「[PDF] 本文の幅を指定する」(p.141)を参照してください。

初心者向け入門書では節と項の見分けをつきやすくする

Re:VIEW では \LaTeX のデザインをそのまま使っているので、節 (Section) と項 (Subsection) のデザインがよく似ており、見分けにくいです (図 6.2)。初心者は本を読み慣れていないので、このデザインでは節と項の違いが分からず、結果として文書構造を理解しないまま読み進めてしまいます。

1.1 節 (Section) タイトル

1.1.1 項 (Subsection) タイトル

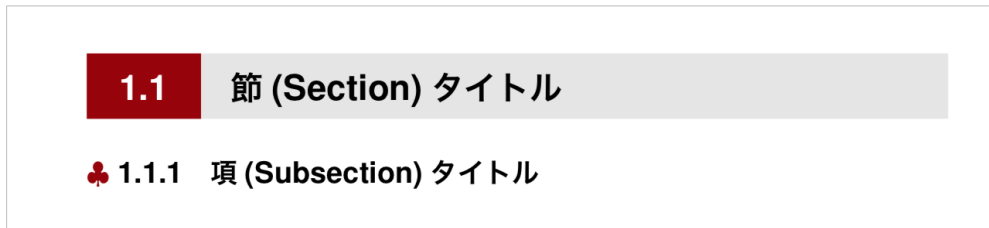
▲ 図 6.2: Re:VIEW では節 (Section) と項 (Subsection) が見分けにくい

Starter では、節と項のデザインを大きく変えており、見分けやすくなっています (図 6.3)。これは、初心者でも節と項の違いがすぐに分かり、文書構造を把握できるようにするための配慮です。

初心者向け入門書では節ごとに改ページする

初心者向けの入門書では、節 (Section) ごとに改ページしていることが多いです。そもそも初心者は本自体を読み慣れていないことが多いので、節ごとに改ページしてあげることで、文章の構造

*1 興味のある人は本屋に行って、入門書とオライリー本の本文を比べてみるといいでしょう。



▲ 図 6.3: Starter では節 (Section) と項 (Subsection) が見分けやすい

を分かりやすく示せます。

Starter では節ごとに改ページする設定が用意されています。詳しくは「[PDF] 節ごとに改ページする」(p.154) を参照してください。

ただし節ごとに改ページすると、当然ですが全体のページ数は増え、それを調整するために本文をあれこれ変更する必要があります。実践するのは時間とお金（印刷代）に余裕がある場合だけにしましょう*2。

*2 逆にいうと、商業の入門書はそれだけのコストを掛けて制作されているということです。

6.3 箇条書き

箇条書きを正しく使い分ける

箇条書きには「番号なし」「番号つき」「ラベルつき」の3つがあります。

番号なしの箇条書きは、項目に順序がないか、あっても重要ではない場合に使います。たとえば次の例では、公開された順に項目が並んでいるもののそこはあまり重要ではなく、項目の順番を入れ替えても文章の意味が成り立つので、番号なしの箇条書きを使います。

▼ サンプル

スタジオジブリ初期の代表作は次の通りです。

- * 風の谷のナウシカ
- * 天空の城ラピュタ
- * となりのトトロ

▼ 表示結果

スタジオジブリ初期の代表作は次の通りです。

- 風の谷のナウシカ
- 天空の城ラピュタ
- となりのトトロ

番号付きの箇条書きは、手順や順位など、項目の順序が重要な場合に使います。たとえば次の例では項目の順序が重要なので、番号付きの箇条書きを使います。

▼ サンプル

スタジオジブリ初期の代表作を、好きな順に並べました。

1. 風の谷のナウシカ
2. 天空の城ラピュタ
3. となりのトトロ

▼ 表示結果

スタジオジブリ初期の代表作を、好きな順に並べました。

1. 風の谷のナウシカ
2. 天空の城ラピュタ
3. となりのトトロ

ラベル付きの箇条書きは、一見すると番号付き箇条書きに似ていますが、数字ではなくアルファ

ベットや文字を使う点が違います。またラベルはあとから参照しやすくするために使い、順番を表すためには使いません。そのため、ラベル付きの箇条書きは「番号なし箇条書きにラベルをつけたもの」として使います。

たとえば次の例では、項目の順序はさほど重要でないので番号なし箇条書きでもいいのですが、あとから項目を参照するのでラベル付き箇条書きにしています。

▼ サンプル

スタジオジブリ初期の代表作は次の通りです。

- (A) 風の谷のナウシカ
- (B) 天空の城ラピュタ
- (C) となりのトトロ

興行収入を調べると、(A)が14.8億円、(B)が11.6億円、(C)が11.7億円でした。

▼ 表示結果

スタジオジブリ初期の代表作は次の通りです。

- (A) 風の谷のナウシカ
- (B) 天空の城ラピュタ
- (C) となりのトトロ

興行収入を調べると、(A) が 14.8 億円、(B) が 11.6 億円、(C) が 11.7 億円でした。

番号付き箇条書きとラベル付き箇条書きは、きちんと使い分けましょう。前者は順序に強い意味がある場合、後者は順序に意味がないまたは弱い場合に使います。

箇条書き直後に継続する段落は字下げしない

段落の途中に箇条書きを入れる場合、箇条書きの直後は字下げ（インデント）をしないようにしましょう。

次の例を見てください。箇条書きの直後に「//noindent」を入れることで、字下げしないようにしています。

▼ サンプル

スタジオジブリ初期の代表作には、

- * 風の谷のナウシカ
- * 天空の城ラピュタ
- * となりのトトロ

//noindent

が挙げられます。

しかしスタジオジブリが発足したのは実はラピュタからであり、ナウシカは厳密にはスタジオジブリの制作ではないのです。

▼ 表示結果

スタジオジブリ初期の代表作には、

- 風の谷のナウシカ
- 天空の城ラピュタ
- とんりのトトロ

が挙げられます。

しかしスタジオジブリが発足したのは実はラピュタからであり、ナウシカは厳密にはスタジオジブリの制作ではないのです。

字下げは段落の始まりを表すために使います。しかし上の例の「が挙げられます。」は段落の始まりではなく途中だと考えられるため、字下げは行いません。

もっとも、段落の途中で箇条書きを入れること自体がよくありません。この例なら、次のように書き換えるといいでしょう。

▼ サンプル

スタジオジブリ初期の代表作には、**次の3つが挙げられます。**

- * 風の谷のナウシカ
- * 天空の城ラピュタ
- * とんりのトトロ

しかしスタジオジブリが発足したのは実はラピュタからであり、ナウシカは厳密にはスタジオジブリの制作ではないのです。

6.4 本文

強調箇所は太字のゴシック体にする

日本語の文章では本文が明朝体の場合、強調箇所は太字にするだけでなく、ゴシック体にするのがよいとされています。そのため、強調には「@{...}」ではなく「@{...}」または「@{...}」を使ってください。

▼ サンプル

本文本文@{強調}本文本文。@<balloon>{ゴシック体なので望ましい}

本文本文@{太字}本文本文。@<balloon>{明朝体のままなので望ましくない}

▼ 表示結果

本文本文**強調**本文本文。←ゴシック体なので望ましい

本文本文**太字**本文本文。←明朝体のままなので望ましくない

強調と傍点を使い分ける

傍点とはこのように文章の上についた小さな点のことであり、Re:VIEW や Starter なら「@<bou>{...}」でつけられます。

傍点は強調とよく似ていますが、強調は「重要な箇所」を表すのに対し、傍点は「重要ではないけど他と区別したい・注意を向けたい」という用途で使います。

たとえば次の例では、否定文であることが見落とされないう、傍点を使っています。ここは重要箇所ではないので、強調は使わないほうがいいでしょう。

Re:VIEW Starter ではアップグレード用スクリプトを用意していません。

また技術系の文書ではほとんど見かけませんが、小説や漫画では「何か含みを持たせた表現」や「のちの伏線になる箇所」に傍点を使います。参考までに、『ワールドトリガー』22 巻^{*3}から傍点を使った例を（ネタバレは避けつつ）引用します。

その^{ハウンド}追尾弾は相手を動かすための^{ハウンド}追尾弾なのよ (p.132)

だとすると、^{あまとり}雨取ちゃんは人を狙って撃てないのか？ (p.133)

^{*3} 『ワールドトリガー』22 巻（葦原大介、集英社、2020 年）

今の千佳^{ちか}は間違いなくちゃんと戦^{いく}う意志を持っていますよ (p.134)

さっきの弾^{たま}は■■■■■を■った■■■■か……!? (p.174)

[PDF] 記号と日本語はくっつくことに注意する

PDF では、読みやすさのために日本語と英数字の間に少しアキ（隙間）が入ります。これは内部で使っている組版用ソフト「 \LaTeX 」の仕様です。

▼ サンプル

あいうえおABC DEFかきくけこ123。

▼ 表示結果

あいうえお ABC DEF かきくけこ 123。

しかし記号の場合はアキが入りません。たとえば次の例では、「ン」と「-」の間にはアキが入っていません。

▼ サンプル

オプション-pを使う。

▼ 表示結果

オプション-pを使う。

このような場合は、半角空白を入れるか、またはカッコでくくるといいでしょう。

▼ サンプル

オプション -p を使う。

オプション「-p」を使う。

▼ 表示結果

オプション -p を使う。

オプション「-p」を使う。

[PDF] 等幅フォントで余計な空白が入るのを防ぐ

Re:VIEW や Starter では、PDF を作成するために「 \LaTeX 」という組版ソフトを使っています。この \LaTeX では、半角空白は次のように扱われます。

- 文章中の半角空白は、何個連続していても1個分の空白しか出力されない。
- ただし「.」と「:」と「!」と「?」の直後の半角空白は、1個しかなくても2個分の空白が出力される。

このルールは、英語の文章を書くうえでは妥当な仕様です。そしてこのルールは、フォントに依らず適用されます。そのため等幅フォントを使っているときでも、「.」と「:」と「!」と「?」の直後に半角空白があると2個分の空白が出力されます。

しかし、等幅フォントを使うのはたいていプログラムコードやコマンドを表す場合なので、半角空白が勝手に2個出力されてしまうのは困りものです。たとえば、

「{a: 1}」と書いたつもりが、

「{a: 1}」と表示されるのは、意図したことではないでしょう。

そこで Starter では、等幅フォントの場合はこのルールを適用しないようにしています。そのため「\frenchspacing」*4というマクロが L^AT_EX にはあるので、これを使います。具体的には、L^AT_EX マクロを次のように上書きしています。

```
%% 「 」を上書き
\renewcommand{\reviewtt}[1]{%
  \frenchspacing%   % 「! ? : .」の直後の空白が2文字分になるのを防ぐ
  \texttt{#1}%       % 等幅フォントで表示
}
```

「@<code>{ }」が呼び出す「\reviewcode」マクロにも同じ変更をしています。このおかげで、Starter では「@<tt>{ }」や「@<code>{ }」を使ったときに空白が2個出力されるのを防いでいます。

ただし、本来このようなルール変更は「@<code>{ }」にだけ行うべきであり、「@<tt>{ }」では変更すべきではありません。現実には「@<tt>{ }」がプログラムコードを表すのに広く使われていることを鑑みて、Starter では「@<tt>{ }」でもルール変更をしています。しかし本来はすべきでないと知っておいてください。

なお「@<tt>{ }」を本来の動作に戻すには、sty/mystyle.sty に次のマクロ定義を追加してください。

▼ sty/mystyle.sty

```
%% @<tt>{ } を本来の動作に戻す
\renewcommand{\reviewtt}[1]{%
  \texttt{#1}%       % 等幅フォントで表示
}
```

*4 「\frenchspacing」というのは、「フランス流の空白の入れ方」という意味です。フランス語では「. : ! ?」の直後でも空白は1個なようです。

文章中のコードは折り返しする

(TODO)

ノートとコラムを使い分ける

ノートとコラムはよく似ています。次のように使い分けるといいでしょう。

- 本文に対する補足情報は、ノートを使う。あくまで補足情報なので、文章が長くなりすぎないように注意する。
- 章 (Chapter) の終わりに、長めの関連情報やエッセーを書く場合はコラムを使う。これは節 (Section) と同じ扱いにするので、「`==[column]`」を使う。

ときどき、ノートで書くべきことをすべてコラムで書いている同人誌を見かけます。あまりよいことではないので、ノートとコラムの使い分けを検討してみてください。また使い分けがよく分からないと思ったら、商業の技術書籍をいくつか参考にしてみてください。

6.5 見出し

節タイトルが複数行になるなら下線や背景色を使わない

Starter では節 (Section) のタイトルに下線や背景色をつけられます。しかしそれらは節タイトルが 1 行に収まる場合だけを想定しており、複数行の場合は考慮されていません (図 6.4)。

複数行になるぐらいの長い節タイトルがある場合は、下線も背景色も使わないデザインを選びましょう。Starter であれば、次のどれかを選ぶといいでしょう。

- 「numbox」… 節番号を白ヌキ文字 (図 6.4 上から 2 番目)
- 「leftline」… 節タイトル左に縦線 (図 6.4 上から 4 番目)
- 「circle」… 大きい円に白ヌキ文字 (図 6.4 上から 5 番目)

このうち「circle」は、節タイトルが 1 行の場合でも 2 行分の高さを必要とするので、他のデザインと比べてページ数が若干増えることに注意してください。

なお Starter における節タイトルのデザインを変更する方法は、「節のタイトルデザインを変更する」(p.151) を参照してください。

項を参照するなら項番号をつける

「@<secref>{ }」や「@<hd>{ }」で項 (Subsection) を参照するなら、項にも番号をつけましょう。番号のついていない見出しを参照するのは止めておいたほうがいいでしょう。

項に番号をつける方法は、「項に番号をつける」(p.155) を参照してください。

「section_decoration: grayback」の場合：

1.1 ぼくのとなりにあんこくはかいしん がいます

「section_decoration: numbox」の場合：

1.1 ぼくのとなりにあんこくはかいしん がいます

「section_decoration: underline」の場合：

1.1 ぼくのとなりにあんこくはかいしん がいます

「section_decoration: leftline」の場合：

1.1 ぼくのとなりにあんこくはかいしん がいます

「section_decoration: circle」の場合：

1.1 ぼくのとなりにあんこくはかいしん がいます

▲ 図 6.4: 節タイトルが長くて複数行になった場合

6.6 プログラムコード

0 と O や 1 と l が見分けやすいフォントを使う

プログラムコードやターミナルでは、0 と O や、1 と l が見分けやすい等幅フォントを使ってください（図 6.5）。具体的には「beramono」フォントか「inconsolata」フォントを使うといいでしょう。

デフォルト（太字にしても目立ちにくい）

```
abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
0123456789 Bold String
"' '(){}[]<>;:,.+-*/%|&|^$?#@~\_\\
```

beramono フォント（太字が目立つ）

```
abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
0123456789 Bold String
"' '(){}[]<>;:,.+-*/%|&|^$?#@~\_\\
```

▲ 図 6.5: 0 と O や、1 と l が見分けやすい等幅フォントを使う

Starter では、プログラムコードやターミナルの等幅フォントを選べる設定を用意しています。詳しくは「[PDF] プログラムコードのフォントを変更する」(p.148) を参照してください。

フォントを小さくしすぎない

プログラムコードやターミナルのフォントサイズが小さすぎると、とても読みづらくなります。具体的には、8pt 以下^{*5}にするのはやめましょう。

昔は長い行をページ内に収める必要があったため、プログラムコードのフォントサイズを小さくする必要がありました。しかし今は長い行を自動的に折り返す機能があるので、小さいフォントを使う必要性は薄れています。また、たとえば「inconsolata」フォントは \LaTeX のデフォルトフォントと比べて小さめに表示されるため、このフォントを選ぶと必要以上に小さく見えます。

^{*5} 念のために説明すると、「8pt 以下」には 8pt も含まれます。

Starter のデフォルトでは、本文のフォントサイズに関わらず、A5 サイズなら 9pt、B5 サイズなら 10pt のフォントサイズが使われます。

- A5 サイズなら、本文が 9pt と 10pt のどちらであっても、プログラムコードはデフォルトで 9pt
- B5 サイズなら、本文が 10pt と 11pt のどちらであっても、プログラムコードはデフォルトで 10pt

この調整は、「config-starter.yml」の設定項目「program_default_options:」と「terminal_default_options:」の中の、「fontsize:」で行われています*6。これらの値は、本文が A5 サイズ 10pt または B5 サイズ 11pt なら「small」に設定され、A5 サイズ 9pt または B5 サイズ 10pt なら「null」に設定されます。

重要箇所を目立たせる

プログラムコードが 5 行以上あると、読者はコードのどこに注目すればいいか、分からなくなります。もし注目してほしい箇所が強調されていれば、読者にとってとても理解しやすくなります。

プログラムコードの中で注目してほしい箇所は、ぜひ太字にして目立たせましょう。たとえば次の例は再帰プログラムの説明なので、再帰している箇所を太字にして目立たせています。

▼ 再帰プログラムの例

```
function fib(n) {
  if (n <= 1) {
    return n;
  } else {
    return fib(n-1) + fib(n-2);
  }
}
```

なおプログラムコードを太字にするときは、「@{ }」ではなく「@{ }」を使ってください。「@{ }」だと等幅フォントのはずがゴシック体が変わってしまいます。「@{ }」だと太字にするだけなので等幅フォントのまま変わりません。

.....

太字では目立たないことがある

太字にしても目立たない文字や記号があるので注意してください*7。

たとえば空白やタブ文字は、目に見えないので太字にしても見えません。またピリオド「.」やカンマ「,」やハイフン「-」やクオート「'」などは、太字にしても目立たず、読者には気づいてもらえない可能性が高いです。

解決策としては、文字の色を変えるか、背景色を変えることです。Starter ではどのような方法にするかを検討中です。

*6 以前は「progoram_fontsize:」と「terminal_fontsize:」という設定項目がありましたが、廃止されました。

*7 重要でない箇所を薄く表示したり、削除したコードに取り消し線をつける場合でも、同様の問題が起こります。

.....

重要でない箇所を目立たせない

Java の「`public static void main(String[] args)`」や、PHP の「`<?php ?>`」は、プログラムコードの説明において重要ではないので、目立たせないようにするといいいでしょう。

Starter では「`@<weak>{...}`」を使うと、プログラムコードの一部を目立たせないようにできます。

▼ 重要でない箇所を目立たせなくした例

```
public class Example {
    public static void main(String[] args) {
        System.out.println("Hello, world!");
    }
}
```

差分（追加と削除）の箇所を明示する

サンプルコードを徐々に改善するような内容の場合は、次のようにすると差分が分かりやすくなります。

- 削除した行に「`@{ }`」で取り消し線を引く。
- 追加した行を「`@<ins>{ }`」で太字にする。

たとえば次のようなサンプルコードがあるとします。

▼ 最初のサンプルコード

```
/// ボタンを押すたびにラベルを入れ替える
$('#button').on('click', function(ev) {
    let $this = $(this);
    let prev = $this.text();
    let next = $this.data('label');
    $this.text(next).data('label', prev);
});
```

これを改善して、次のようなコードにしました。しかしこれだと、どこを変更したかを読者が探さないといけません。

▼ 改善したサンプルコード

```
/// ボタンを押すたびにラベルを入れ替える
function toggleLabel(ev) {
    let $this = $(this);
    let prev = $this.text();
    let next = $this.data('label');
```

```

    $this.text(next).data('label', prev);
}
$('#button').on('click', toggleLabel);

```

もし次のように差分を分かりやすく表示すれば、読者は変更点をすぐに理解できるでしょう。

```

/// ボタンを押すたびにラベルを入れ替える
$('#button').on('click', function(ev){
function toggleLabel(ev) {
    let $this = $(this);
    let prev = $this.text();
    let next = $this.data('label');
    $this.text(next).data('label', prev);
}
});
}
$('#button').on('click', toggleLabel);

```

この作業はとても面倒です。それでもこの面倒を乗り切った本だけが、初心者にとって分かりやすい本になるのです。

このほか、Git での差分のような表示方法も考えられます。各自で工夫してみてください。

長い行の折り返し箇所を明示する

長い行が右端で折り返されるとき、折り返された箇所が分かるような表示にするといいでしょう。そのための方法は3つあります。

- 折り返し箇所に何らかの目印をつける
- 行番号をつける
- 行末を表す記号をつける

たとえば次の例では、あたかも複数行あるかのように見えます。

```

*****
*****
*****

```

しかし折り返した箇所に折り返し記号をつけてみると、実は1行であることが分かります。

```

*****>
>*****>
>*****>

```

また行番号をつけても、1行であることが分かります。

```
1: *****
*****
*****
```

あるいは行末を表す記号をつける方法でも、折り返した行にはその記号がつかないので、1行であることが分かります。

```
*****
*****
*****
```

このように、方法は何でもいいので折り返し箇所が分かるような仕組みを使いましょう。

Starter では、デフォルトで折り返し箇所に折り返し記号がつくようになってます。また折り返し記号がプログラムコードの一部だと間違えて認識されないよう、次のような工夫をしています。

- フォントの色をグレーにして薄く表示する。
- フォントの種類を等幅でなくローマン体にする。

行番号を控えめに表示する

行番号は、プログラムコードと同じフォント・同じ色で表示すべきではありません。行番号はあくまで脇役なので、主役であるプログラムコードよりも目立たないようにすべきです。

Starter では、行番号をグレー表示しているのので、行番号がプログラムコードよりも目立たません。

```
1: function fib(n) {
2:     if (n <= 1) {
3:         return n;
4:     } else {
5:         return fib(n-1) + fib(n-2);
6:     }
7: }
```

行番号を考慮して長い行を折り返す

長い行を自動的に折り返すとき、もし行番号があればそれを考慮した表示にすべきです。

Starter ではそのような表示になっています。

```
1: if error:
2:     sys.err.write("Something error raised. Please contact to system administrator.\n")
```

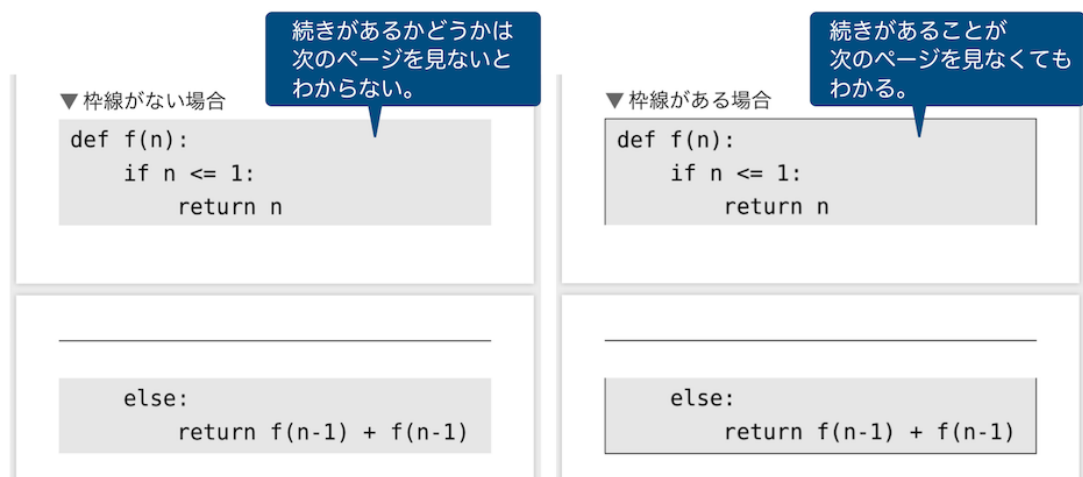
この例を見ると、折り返した行が左端には行かず、行番号の表示スペースを避けて折り返されています。これが、行番号つき折り返しの望ましい姿です。

これに対し、Re:VIEW では折り返した行が左端に到達してしまい、行番号の表示スペースを侵食します。これは行番号つき折り返しの望ましくない姿です。

ページまたぎしていることを可視化する

プログラムコードがページまたぎしていることが分かるようにしましょう。

- プログラムコードに枠線がないと、ページまたぎしたときに次のページに続いているかどうか、次のページを見ないと分かりません（図 6.6 左）。
- プログラムコードに枠線があれば、ページまたぎしても次のページに続いているかどうか、次のページを見なくても分かります（図 6.6 右）。



▲ 図 6.6: プログラムコードに枠線がない場合とある場合の違い

Starter ではデフォルトでプログラムコードに枠線がつきます。枠線をつけたくない場合は「config-starter.yml」の設定項目「program_border:」を「false」に設定します。

なお、プログラムコードの背景枠の四隅を角丸にすることでも、ページまたぎしていることを可視化できます。

インデントを可視化する

Python や YAML ではインデントでブロック構造を表すので、ブロックの終わりを表すキーワードや記号がありません。そのため、プログラムコードがページをまたぐとインデントが分からなくなる（つまりブロックの構造が分からなくなる）ことがあります。

これを防ぐには、何らかの方法でインデントを可視化します。そうすると、プログラムコードがページをまたいでもインデントが分からなくなることはありません。

Starter では「`//list[][[indent=4]{ ... //}`」^{*8}のようにすることで、インデントごとに

^{*8} 後方互換性のために、「indentwidth=4」という名前でも指定できます。

薄い縦線をつけられます。

▼ 薄い縦線をつけてインデントを可視化する

```
class Fib:

    def __call__(n):
        if n <= 1:
            return n
        else:
            return fib(n-1) + fib(n-2)
```

6.7 図表

図表が次のページに送られてもスペースを空けない

Re:VIEW ではデフォルトで、図は現在位置に置かれるよう強制されます。もし現在位置に図を入れるスペースがない場合は、図は次のページに表示され、そのあとに本文が続きます。この仕様のせいで、たくさんのスペースが空いて本文がスカスカになってしまうことがあります（図 6.7 上）。

これはよくないので、Starter では図が次のページに送られた場合、後続の本文を現在位置に流し込みます。図が現在位置に入らないからといって、スペースを空けたままにする必要はありません（図 6.7 下）。

大きい図表は独立したページに表示する

たとえば図がページの 3/4 を占めるようなら、そのページには後続のテキストを流しこまず、その図だけのページにしましょう。

Starter だと「`//image[ファイル名][説明文][scale=1.0,pos=p]`」とすると、その図だけの独立したページに表示されます。

図表は番号で参照する

図や表は、現在位置に入り切らないとき、自動的に次のページに送られます。その場合、たとえば「次の図を見てください」と書いてあると「次の図」がなくて読者が混乱します。

そのため、図や表は必ず番号で参照してください。図なら「`@{ファイル名}`」、表なら「`@<table>{ラベル}`」で参照します。

白黒印刷でも問題ないか確認する

形は同じで色だけが違う図形を使うと、白黒印刷したときに判別がつかなくなります。

白黒印刷するなら、形を変えたり線の種類を変えたりするなどして、白黒印刷しても判別できるような図にしましょう。

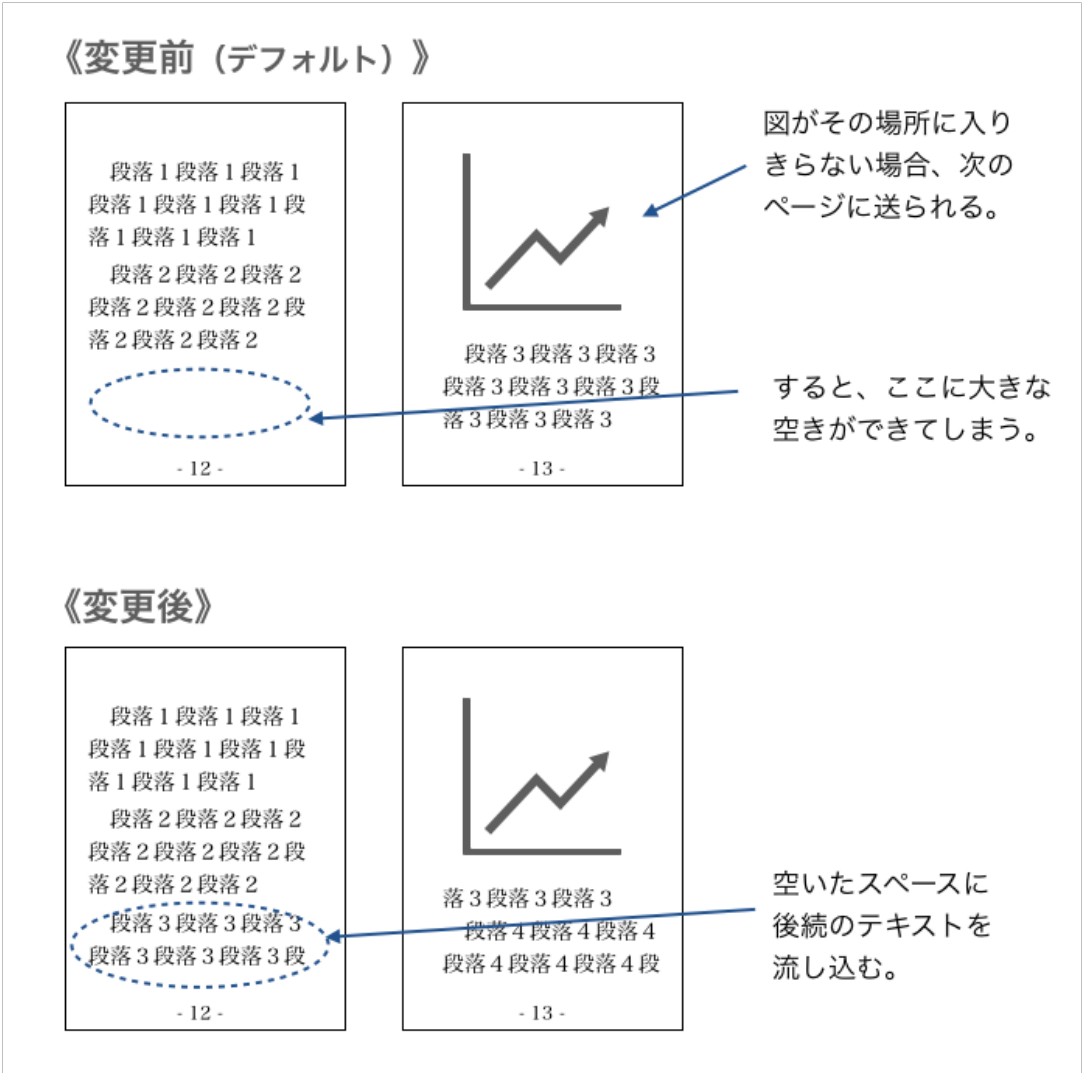
表のカラム幅を指定する

表に長い文章を入れると、ページ横にはみ出してしまいます。

このような場合は、「`//tsize[latex][p{70mm}]`」のようにカラムの幅を指定しましょう。

▼ サンプル

```
//tsize[latex][|l|p{70mm}||]
//table[tbl-jquq9][長いテキストを使ったサンプル]{
伝承地          伝承内容
-----
```



▲ 図 6.7: 後続の本文を現在位置に流し込むかどうか

風の谷 その者、蒼き衣を纏て金色の野に降り立つべし。失われし大地との絆を
結び、ついに人々を青き清浄の地に導かん。
 Gondar no Tani Rite・Ratbarita・Urus・Ariarosu・Baru・Nettiru
 // }

▼ 表示結果

表の列が数値なら右寄せにする

表の列はデフォルトで左寄せ (l) ですが、右寄せ (r) や中央揃え (c) を指定できます。特に

▼ 表 6.1: 長いテキストを使ったサンプル

伝承地	伝承内容
風の谷	その者、蒼き衣を纏て金色の野に降り立つべし。失われし大地との絆を結び、ついに人々を青き清浄の地に導かん。
Gondaria の谷	リーテ・ラトバリタ・ウルス・アリアロス・バル・ネトリー

カラムが数値なら、右寄せにするのがいいでしょう。

▼ サンプル

```
//tsize[latex][|l|c|r|]
//table[tbl-o599k][左寄せ・中央揃え・右寄せのサンプル]{
左寄せ          中央揃え      右寄せ
-----
1                1              1
10               10             10
100              100            100
1000             1000           1000
//}
```

▼ 表示結果

▼ 表 6.2: 左寄せ・中央揃え・右寄せのサンプル

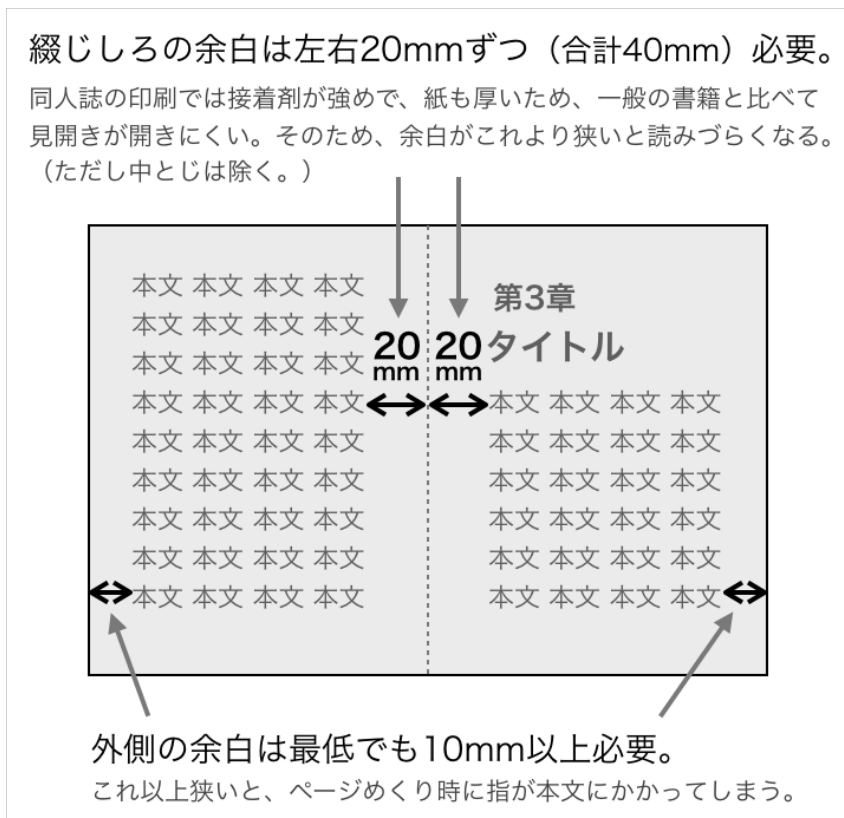
左寄せ	中央揃え	右寄せ
1	1	1
10	10	10
100	100	100
1000	1000	1000

6.8 ページデザイン

[PDF] 印刷用では左右の余白幅を充分にとる

紙の書籍では、左右いっばいに本文を印刷してはいけません。必ずページ左右の余白を充分に確保してください（図 6.8）。

- 見開きにおいて内側の余白（つまり左ページの右余白と、右ページの左余白）は、最低 20mm が必要です。それ以上切り詰めると、本を開いたときに内側の本文がとても読みにくくなります。
- 見開きにおいて外側の余白（つまり左ページの左余白と、右ページの右余白）は、10～15mm くらいが必要です。それ以上切り詰めると、指が本文にかかってしまうので読みにくくなります。



▲ 図 6.8: 紙の書籍ではページ左右の余白幅が必要

Starter ではこのような制限を考慮しつつ、本文幅が最大になるよう設定しています。そのため、**印刷用 PDF ファイルでは奇数ページと偶数ページで左右の余白幅が違います**。これは意図した仕様でありバグではありません。

電子用 PDF ファイルではこのようなことを考慮する必要はありません。そのため、電子用 PDF では左右の余白幅は同じです。

[PDF] タブレット向けでは余白を切り詰める

Starter では、印刷用 PDF と電子用 PDF を切り替えて生成できます。このとき、電子用 PDF のレイアウトは印刷用 PDF をほぼ忠実に再現しています。つまり電子用 PDF のレイアウトは電子用として最適化されているわけではなく、あくまで印刷用のレイアウトを流用しているだけです。

もし印刷用 PDF を必要とせず、電子用 PDF だけを必要とするなら、レイアウトをタブレット向けに最適化するといいいでしょう。タブレット向けの PDF では、本文周辺の余白を切り詰めると、7 インチや 8 インチのタブレットでの読みやすさが向上します。余白がまったくないのもどうかと思いますが、全角 1 文字分ぐらいの余白があればタブレットの画面では十分です。

Starter では、プロジェクト作成時にタブレット向けのレイアウトを選択できます。そうすると、タブレット向けに余白を切り詰めたレイアウトになります。印刷用 PDF を必要としない人は試してみてください。

なおタブレット向けに余白を切り詰めた A5 サイズの本文幅は、切り詰めていない通常の B5 サイズの本文幅とほぼ同じになります。つまり B5 サイズで余白を切り詰めると本文幅が長くなりすぎるので、Starter ではタブレット向けとしては A5 サイズのみを用意しています。

[PDF] 電子用 PDF ではページ番号の位置を揃える

紙の書籍においてページ番号は、見開きで左右の上隅または下隅に置かれることが多いです。紙の書籍は見開きで読めるので、ページ番号がこのような位置でも問題ありません。

しかしこれと同じことを電子用 PDF で行くと、ページ番号の位置が右上だったり左上だったり（あるいは右下だったり左下だったり）一定しないので、読みづらいです。なぜなら、電子用 PDF をタブレットで見るときは見開きではなく 1 ページずつ読まれるからです。

つまり紙の書籍におけるページ番号は見開きであることを前提にしており、それと同じ前提を電子用 PDF にしてはいけません。

電子用 PDF では、ページ番号の位置は一定であることが望ましいです。そのため、Starter ではページ番号をページ下の真ん中に置いています。またこの位置は紙の書籍でも通用するので、印刷用 PDF と電子用 PDF とで同じページレイアウトにできます。

非 Retina 向けには本文をゴシック体にする

Re:VIEW や Starter では、本文のフォントを明朝体になっています。明朝体は、印刷物や Retina ディスプレイのように解像度が高いと読みやすいですが、Retina でない（つまり高解像度でない）ディスプレイだと読みづらいです。

スマートフォンやノートパソコンでは Retina ディスプレイが普及しましたが、外部ディスプレイでは高解像度ではないものがまだ一般的です。

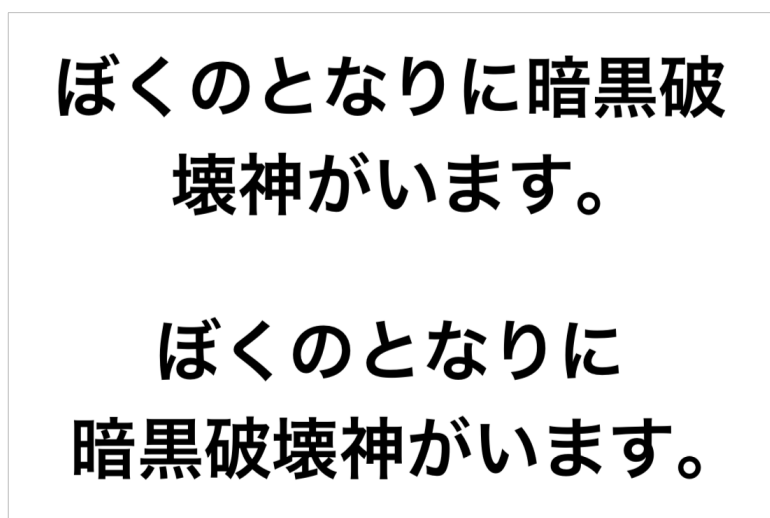
もし高解像度でないディスプレイでも読みやすくしたいなら、本文をゴシック体にする 것을検討しましょう。Starter では「**config-starter.yml**」の設定を変えるだけで変更できます。詳しくは「[PDF] フォントの種類を変更する」(p.145)を参照してください。

6.9 大扉

長いタイトルでは改行箇所を明示する

本や同人誌のタイトルが長いと、大扉（タイトルページ）において不自然な箇所で改行されてしまいます（図 6.9 上）。

これを防ぐために、Starter ではタイトルを複数行で指定できるようになっています。この場合、大扉（タイトルページ）ではタイトルが 1 行ずつ表示されるので、自然な位置を指定して改行できます（図 6.9 下）。



▲ 図 6.9: タイトルの改行位置を指定しなかった場合（上）とした場合（下）

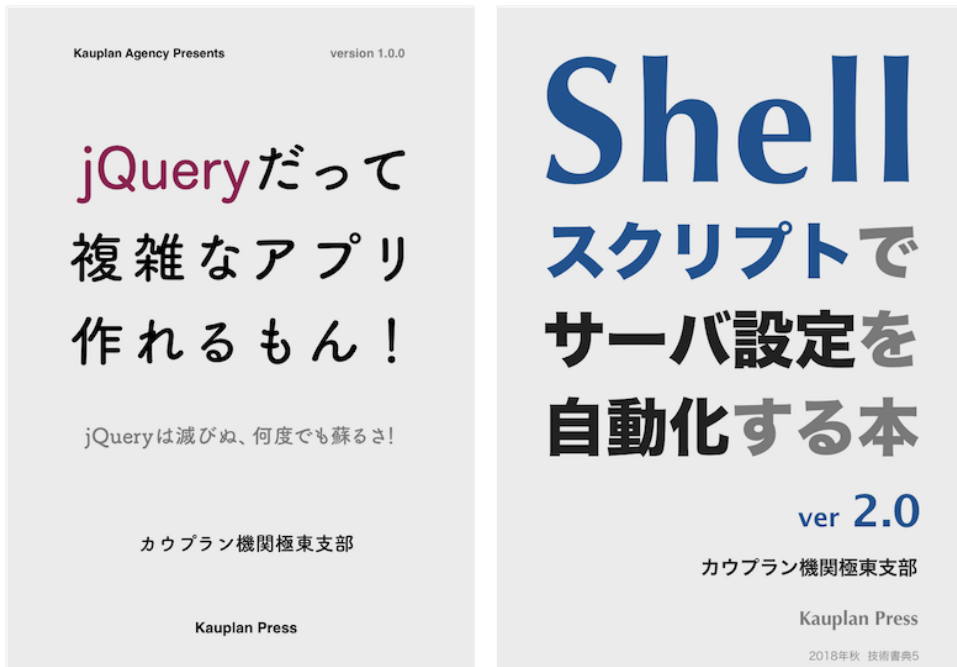
大扉を別のソフトで作成する

Re:VIEW や Starter が生成する大扉（タイトルページ）は、デザインがよくありません。Photoshop や Illustrator や Keynote で作成したほうが、見栄えのいいデザインになります（図 6.10）。

Starter では、別のソフトで作成した大扉や奥付の PDF ファイルを読み込めます。やり方は「[PDF] 表紙や大扉や奥付となる PDF ファイルを指定する」（p.157）を参照してください。

またこうして読み込んだ PDF ファイルにも、ノンブルがつきます。以前は大扉や奥付の PDF ファイルを読み込む機能がなかったため、最初にノンブルなしの印刷用 PDF を生成し、大扉や奥付を差し替えてから、最後にノンブルをつける必要がありました。今は印刷用 PDF 生成時に大扉や奥付の PDF ファイルを読み込めるので、最初からノンブルつきで印刷用 PDF が生成できます。

このように、別ソフトで作成した大扉を取り込む機能が Starter では整っています。積極的に別ソフトで作っていきましょう。なお印刷所へ入稿するなら、大扉に背景色をつける場合は塗り足



▲ 図 6.10: Keynote で作成した大扉の例

し^{*9}もつけましょう。「塗り足し」が分からなければ、大扉の背景色を白にしておくと入稿時のトラブルが少なくなります。

.....

大扉を Keynote や Pages で作成する

Keynote.app で大扉を作成しようとする、困ったことに Keynote.app にはスライドのサイズを A5 や B5 にする機能がありません。かわりにスライドを次のようなカスタムサイズにすると、ちょうど A5 や B5 のサイズになります。

- A5 の場合：419pt x 595pt
- B5 の場合：516pt x 728pt

また Pages.app なら簡単に A5 や B5 にできます。B5 にしたい場合は「JIS B5」を選んでください。

.....

^{*9} 塗り足しについては「[PDF] 塗り足しをつける」(p.142)を参照してください。

6.10 奥付

奥付は最後のページに置く

紙の本は必ず偶数ページになります。たとえば、ページ数が 100 ページの本はあっても 101 ページの本はありません。そのため、本の最終ページも偶数ページになります。

そして奥付は、本の最後のページに置きます。言い換えると、奥付は本の最後の偶数ページに置かれるはずです。しかし、このことが考慮されていない同人誌がたまにあります。

実は素の Re:VIEW ではこれが考慮されておらず、奥付が奇数ページに置かれることがあります*10。Starter ではこのようなことはありません。奥付は必ず最後の偶数ページに置かれます。気が利いてますね。

奥付に更新履歴とイベント名を記載する

一般的に、奥付には本の更新履歴が記載されます。

同人誌ならそれに加えて、初出のイベント名も記載するといいいでしょう。そうすると、どのイベントで手に入れたかが分かりやすくなります*11。

利用した素材の作者と URL を奥付に記載する

本の作成に使用した素材があれば、作者名と URL を奥付に記載するといいいでしょう。

たとえば表紙に写真やイラストを使ったのであれば、その作者と素材への URL を奥付に記載しましょう。

利用したソフトウェアを奥付に記載する

本や同人誌をどんなソフトウェアで作ったかを奥付に記載すると、他の人の参考になります。

たとえば「InDesign で作った」「MS Word 2016 で作った」「Pages 10.0 で作った」「Vivliostyle で作った」という情報があると、これから本や同人誌を作ろうとする人にとって大きな手がかりとなります。

またエディタに強いこだわりがある人なら「VS Code で執筆した」「Vim で執筆した」のようにアピールするのもいいでしょう。

*10 TechBooster のテンプレートでは偶数ページに置かれるように修正されています。

*11 少なくとも新刊はこれが当てはまります。既刊だと当てはまりませんが、初出のイベント名が分かるだけでも読者の助けになります。

6.11 textlint による文章チェック

この章では、「textlint」というツールを使って文章をチェックする方法を紹介します。

textlint とは

「textlint^{*12}」とは、日本語の文章をチェックして改善点を指摘してくれるツールです。

たとえば、次のような内容のファイルがあるとします。ファイル名の拡張子は「.rd」ではなく「.md」であることに注意してください。

▼ sample1.md : 改善前

Re:VIEW Starterを使うと、きれいなPDFファイルを作成することができます。

これを textlint でチェックすると、「もっと簡潔に書けるよ」と指摘してくれます（textlint のインストールについてはあとで説明します）。

```
$ textlint sample1.md

/Users/yourname/sample1.md
  1:35 ✓ error 【dict2】 "することができます"は冗長な表現です。"することが"を省く
  き簡潔な表現にすると文章が明瞭になります。
  解説: https://github.com/textlint-ja/textlint-rule-ja-no-redundant-expression#dic
  t2 ja-technical-writing/ja-no-redundant-expression

☒ 1 problem (1 error, 0 warnings)
✓ 1 fixable problem.
Try to run: $ textlint --fix [file]
```

指摘された通りに改善してみましょう。

▼ sample1.md : 改善後

Re:VIEW Starterを使うと、きれいなPDFファイルを作成することができます。

これを textlint でチェックすると、指摘されなくなりました。

```
$ textlint sample1.md ←何もエラーが出ない
```

もう1つ、別の例を見てみましょう。たとえば次のような文章があるとします。

▼ sample2.md : 改善前

^{*12} <https://github.com/textlint/textlint>

ノンブルは、印刷所に入稿するときに必要です。

一見すると問題がない文章ですが、textlint でチェックすると「1 つの文に複数の『に』があるよ」と指摘してくれます。

```
$ textlint sample2.md

/Users/yourname/sample1.md
  1:17 error 一文に二回以上利用されている助詞 "に" がみつかりました。  japanese/n>
o-doubled-joshi

☒ 1 problem (1 error, 0 warnings)
```

指摘されたことを改善してみましょう。

▼ sample2.md : 改善後

ノンブルは、印刷所へ入稿するときに必要です。

改善後は、指摘されなくなりました。

```
$ textlint sample2.md ←何もエラーが出ない
```

このように、textlint を使うと日本語文章の改善できそうな点が分かります。とはいえ、textlint も万能ではありません。次の項で注意点を説明します。

textlint の注意点

textlint は似たような文章であっても、改善点を指摘してくれないことがあります。たとえば先ほどの 1 番目の例である「作成することができます」だと改善点を指摘してくれますが、「作ることができます」だと指摘してくれません。前者が「作成できます」に改善できたように、後者も「作れます」に改善できます。

また textlint は「どこが改善できるか」を教えてくださいますが、「どう改善できるか」は必ずしも教えてくれません。先ほどの 2 番目の例がまさにそれで、『に』が複数あることは教えてくれますが、どう直せばいいかは自分で考える必要があります。

さらに textlint の指摘がいつも正しいわけではありません。たとえば「血も涙もない。」という文章を textlint でチェックすると、「1 つの文章に複数の『も』があるよ」と指摘されます。しかしこの指摘がおかしいことは、明らかでしょう。

▼ sample3.md : 修正前

血も涙もない。

```
$ textlint sample3.md

/Users/yourname/sample3.md
  1:4  error  一文に二回以上利用されている助詞 "も" がみつかりました。  japanese/no-
>-doubled-joshi

☒ 1 problem (1 error, 0 warnings)
```

『も』が複数個現れないように修正できますが、これはさすがにやりすぎです。しないほうがいいでしょう。

▼ sample3.md：修正後

```
血も涙もない。
血と涙のどちらも欠いている。
```

```
$ textlint sample3.md  ←何もエラーが出ない
```

このような過剰な指摘が起きたのは、Starter がデフォルトで用意した textlint 用の設定が、技術文章向けだからです。textlint は、事前に設定されたルールに従って文章をチェックするツールです。設定されたルールが小説向けであれば小説向けのチェックをし、技術文章向けであれば技術文章向けのチェックをします。

textlint の設定は、プロジェクトフォルダの直下にある「.textlintrc」というファイルで変更できます。たとえば次のように変更すると、複数の『も』が登場してもエラーにならなくなります。

▼ .textlintrc

```
{
  "filters": {
  }
, "rules": {
  "preset-japanese": false
, "preset-ja-technical-writing": true
, "preset-ja-technical-writing": {
  "no-doubled-joshi": {
    "allow": ["も"]
  }
}
, "preset-jtf-style": false
, "preset-ja-spacing": false
, "preset-ja-engineering-paper": false
}
```

「.textlintrc」についてこれ以上の解説はしません。詳しく知りたい人は「textlint 使い方」で Google 検索してください。

ここまでが、textlint についての一般的な説明でした。次の項から、Starter で textlint を使う方法を説明します。

Starter のプロジェクトで textlint を使う

Starter のプロジェクトで textlint を使うには、次のようにします。

- (1) Node.js と npm コマンドをインストールする。
 - (2) textlint をインストールする。
 - (3) textlint で原稿をチェックする。
- (1) と (2) は準備なので、プロジェクトごとに一度だけ行います。
それぞれ順番に説明します。

(1) Node.js と npm コマンドをインストール

textlint を実行するには、Node.js^{*13}と npm コマンドをインストールする必要があります。

- Docker を使っている場合は、Docker イメージの中に Node.js と npm コマンドがインストール済みなので、何もする必要がありません。
- Docker を使っていない場合は、<https://nodejs.org/>にアクセスして Node.js をインストールしてください。npm コマンドも自動的にインストールされます。

.....

Node.js のバージョンについて

インストールする Node.js のバージョンは、「LTS」と書かれたほうを選ぶといいでしょう。「Current」と書かれたほうのバージョンでも動作しますが、こちらは上級者向けです。

.....

(2) textlint をインストール

次に、textlint とその関連ライブラリをインストールします。Starter ではそのための Rake タスクが用意されているので、次のコマンドを実行してください。

▼ textlint と関連ライブラリをインストール

```
$ rake textlint:setup      ← Dockerを使わない場合
$ rake docker:textlint:setup ← Dockerを使う場合
```

これで textlint のインストールは完了しました。

(3) textlint で原稿をチェック

textlint を使って原稿の文章をチェックするには、「rake textlint」（または「rake

^{*13} <https://nodejs.org/>

`docker:textlint`」) を実行してください。

▼ textlint で原稿の文章をチェックする (結果は省略)

```
$ rake textlint           ← Dockerを使わない場合
$ rake docker:textlint    ← Dockerを使う場合
```

たくさんのエラーが出るので驚くでしょう。指摘された点のすべてを修正する必要はありません。自分で納得できたことだけを修正すればいいです。

ただし出力結果を見ると、ファイル名と行番号が原稿とは違うことに気づくはずです。この理由を次の項で説明します。

Starter における textlint 実行の仕組み

textlint は、Starter の原稿ファイル (*.re) には対応していません。そこで Starter では、次のような仕組みで textlint を実行します。

- (1) 原稿ファイル (*.re) を Markdown 形式のファイル (*.md) に変換する。
- (2) Markdown 形式のファイルを textlint でチェックする。

このような仕組みのため、textlint が報告するのは Markdown 形式のファイルにおける行番号であり、原稿ファイルの行番号ではありません。よって textlint の指摘事項を反映するには、出力された行番号をもとに Markdown ファイルを見て指摘内容を確認し、原稿ファイルに戻って修正するという手順になります。

これは理想的とはいえないです。解決方法は3つ考えられます。

- (A) textlint を改造して、Starter の原稿ファイルを直接読んで解釈できるようにする。
- (B) 原稿ファイルと Markdown ファイルの行番号が一致するように変換する。
- (C) 原稿ファイルと Markdown ファイルとの行番号のマッピングデータを生成し、それを利用して textlint の出力結果に含まれる行番号を Markdown ファイルの行番号から原稿ファイルの行番号に書き換える。

どれも手間のかかる作業になるので、将来も含めて対応はあまり期待しないでください。もしもっといい方法を思いついた人がいたら、ぜひ教えてください。

Markdown への変換は精度がよくない

Starter における Markdown 形式への変換機能は textlint で使うために実装したので、「textlint で使えればそれでいい」という割り切りをしています。そのため、Markdown 形式への変換は精度があまりよくありません。

「こう改善してほしい」と具体的に提案していただければ対応しますが、そもそもあまり力を入れた機能ではないことをご承ください。

付録 A

Re:VIEW との差分

Starter における機能追加や修正の一覧です。

A.1 まとめ記事

こちらの記事も参照してください。

- 『技術系同人誌を書く人の味方「Re:VIEW Starter」の紹介』(Qitta)
<https://qiita.com/kauplan/items/d01e6e39a05be0b908a1>
- 『Re:VIEW Starter の新機能 (2019 年夏)』(Qitta)
<https://qiita.com/kauplan/items/dd8dd0f4a6e0eb539a98>
- 『Re:VIEW Starter の新機能 (2019 年冬)』(Qitta)
<https://qiita.com/kauplan/items/e36edd7900498e231aaf>

A.2 プロジェクト作成

- Starter では、Web サイトにおいて GUI でプロジェクトの作成と初期設定ができます。このおかげで、初心者でもつまづかずにプロジェクトを始められます。

Re:VIEW ではプロジェクトの作成はコマンドラインでしか行えず、また初期設定を自分で行う必要があるので、初心者にはかなりハードルが高いです。

- Starter では、ダウンロードしたプロジェクトに文章のサンプルが含まれています。そのおかげで、どのように原稿を書けばいいのかが分かります。

Re:VIEW では、作成したプロジェクトの中身はほとんど空であり、サンプルとしては役に立ちません。

A.3 コメント

- Starter では、範囲コメントが使えます。

Re:VIEW では、行コメントしか使えません。

- Starter では、行コメントは単に読み飛ばされます。

Re:VIEW では、行コメントが空行扱いになるため、行コメントを使うと意図せず段落が区切られます。これは明らかに Re:VIEW の仕様バグですが、Re:VIEW 開発チームはこのバグを認めていません。

A.4 箇条書き

- Starter では、順序つきリスト（HTML でいうところの「」）の項目に、「1.」や「(a)」や「(A-1)」など任意の文字列が使えます。
Re:VIEW では「1.」しか使えません。
- Starter では、順序つきリストを入れ子にできます。
Re:VIEW ではできません。
- Starter では、順序つきリストと順序なしリストを相互に入れ子にできます。
Re:VIEW ではできません。
- Re:VIEW では、箇条書きの項目が複数行のテキストだと、自動的に 1 行に連結されます。そのせいで、本来なら「aaa」と「bbb」と「ccc」の 3 行に分かれていた単語が、「aaabbbccc」のように勝手に連結されてしまいます。
Starter ではこのようなおかしいバグは起こりません。

A.5 定義リスト

- Re:VIEW では、定義リストの説明文が複数行だと、自動的に 1 行に連結されます。そのせいで、本来なら「aaa」と「bbb」と「ccc」の 3 行に分かれていた単語が、「aaabbbccc」のように勝手に連結されてしまいます。

Starter ではこのようなおかしいバグは起こりません。

- Starter では、定義リストの説明文に箇条書きが書けます。

Re:VIEW では書けません。

- Starter では、定義リストとよく似た「説明リスト」が用意されています。説明リストは入れ子のブロック命令で書くので、説明文の中にたとえばプログラムコードを入れられます。またブロック命令にオプション引数を指定すると、太字にする・しないや、コンパクトモードにする・しないが選べます。

A.6 見出し（章、節、項、目）

- Starter では、章や節のタイトルを見栄えのいいデザインにしています。また設定ファイルを変更することで、デザインを選択できます。

Re:VIEW では L^AT_EX のデフォルトのデザインを使っているだけです。

- Starter では、章 (Chapter) ごとにタイトルページを作成できます。これは商業誌では一般的なスタイルです。
- Starter では、節 (Section) ごとに改ページできます。これは初心者向け入門書ではよく見かけるレイアウトです。
- Starter では、章や節を参照するとページ番号もつきます。これは特に、項 (Subsection) に番号をつけない設定において、項を参照するときに有用です。

A.7 インライン命令

- Starter では、たとえば「`@<code>{foo@{bar}}`」のようにインライン命令を入れ子にできます。

Re:VIEW ではできません。

- Starter では、「`@{}`」が太字かつゴシック体になります。なぜなら日本語の文章では、強調は太字にするだけでなくゴシック体にすることが望ましいからです。

Re:VIEW では、「`@{}`」は太字になるだけです。

- Starter では、「`@{}`」の省略形として「`@{}`」を用意しています。これにより、文章の強調が簡単にできます。

Re:VIEW には、「`@{}`」はありません。

- Starter では、何もしない命令「`@<nop>{}`」があります。これを使うと、たとえば「`@<nop>{{}}`」とすれば「`@{}`」と表示されます。これは Re:VIEW を使って Re:VIEW のマニュアルを作るときに重宝します。

Re:VIEW にはこのようなコマンドはありません。

- Re:VIEW では、たとえば「`@<code>{a: b? c! d. e}`」と入力すると、「`{a: b? c! d. e}`」のように半角空白が 2 つ分出力されます。このような意図しない出力になるのは、 \LaTeX におけるスペーシングの仕様をそのまま引き継いでいるからです。

Starter ではこの現象を回避しており、等幅フォントでも半角空白が 2 つ分出力されないように修正しています。

- Starter では、ファイル名を表すインライン命令「`@<file>{}`」が用意されています。
- Starter では、文字サイズを変更するインライン命令「`@<small>{}`」「`@<xsmall>{}`」「`@<xxsmall>{}`」「`@<large>{}`」「`@<xlarge>{}`」「`@<xxlarge>{}`」が用意されています。
- Starter では、「 \LaTeX 」を表示する「`@<LaTeX>{}`」、「 \TeX 」を表示する「`@<TeX>{}`」、「 \heartsuit 」を表示する「`@<hearts>{}`」が用意されています。
- Starter では、リンクの URL を自動的に脚注に表示する機能があります。これは、紙の本ではリンクの URL が表示されない問題に対する解決策です。

A.8 ブロック命令

- Starter では、ブロック命令を入れ子にできます。たとえば「`//note{ ... //}`」の中に「`//list{ .. //}`」や「`//quote{ ... //}`」を入れられます。
- Starter では、章の概要を表す「`//abstract`」が用意されています。

Re:VIEW にはありませんが、かわりにリード文を表す「`//lead`」が使えます。

- Starter では、縦方向の空きを入れる「`//vspace`」が用意されています。マイナスの値を指定すれば、余分な空きを削除できます。
- Starter では、縦方向のスペースを確保する「`//needvspace`」が用意されています。

たとえばプログラムのキャプションだけが現在のページ、プログラムコードは次のページに表示されてしまったとします。このとき「`//list`」の前に「`//needvspace[6zw]`」と書くと、全角 6 文字分の高さのスペースがなければ自動的に改ページされます（つまりプログラムのキャプションの前で改ページされる）。結果として、プログラムのキャプションとコードが同じページに表示されます。

A.9 プログラムコード

- Starter では、「1」と「l」、「0」と「O」の見分けが付きやすいフォントを選んでいいます。

Re:VIEW では \LaTeX のデフォルトを使っているため、これらの見分けが付きにくいのです。

- プログラムコードを表示するとき、Starter では1つのブロック命令「`//list`」だけで済みます。

Re:VIEW では「`//list`」「`//emlist`」「`//listnum`」「`//emlistnum`」の4つを使い分ける必要があります。

- Starter では、長すぎる行を自動的に右端で折り返します。また折り返したことが分かるような記号をつけてくれます。

Re:VIEW にはこのような気の利いた機能がありません。

- Starter では、「`@{ }`」で取り消し線を引いた行でも右端で折り返しされます。Starter ではこのような細かいところに配慮が行き届いています。

- Starter では、行番号は目立たないようにグレーで表示します。

Re:VIEW にはこのような気の利いた機能がありません。

- Starter では、行番号を欄外に表示できます。これにより、行番号の分だけコードの表示範囲が狭まってしまうのを防げます。

- Starter では、飛び飛びの行番号を指定できます。

Re:VIEW では連続した行番号しか表示できません。

- Starter では、インデントを可視化できます。この機能は、Python のようなインデントで入れ子表現するプログラミング言語では重要です。

- Starter では、ラベルさえ指定されていればキャプションがなくても番号つきで表示されます。

Re:VIEW では、「`//list[fib1]`」のようにラベルを指定していても、キャプションがなければ番号が付きません。ラベルの目的を考えれば、これは Re:VIEW の仕様バグです。

- Starter では、「`//list[?]`」のようにすると簡単に番号付きで表示されます。

Re:VIEW にはこの機能がないので、番号つきで表示するには重複しないラベルを指定する必要があります、数が多いとかなり面倒です。

- Starter では、プログラムコードのフォントサイズを変更できます。「あるプログラムコードだけフォントサイズを小さくする」といったことが簡単にできます。

- Starter では、コード中の注釈を表す「`@<balloon>{ }`」がグレー表示されます。これは注釈がコードより目立たないようにという配慮です。

Re:VIEW にはこのような配慮はありません。

- プログラム中にタブ文字がある場合、Re:VIEW では空白文字に展開されます。しかしこれが「 \LaTeX コードに変換してからタブ文字を展開する」という仕様のため、表示が崩れます。

Starter ではタブ文字を展開してから \LaTeX コードに変換する」ため、表示が崩れません

(ただし限界はあります)。

- Starter では、全角文字の幅を半角文字 2 つ分に揃えて表示する機能があります。全角と半角が混在しているせいで表示が揃わない（崩れてしまう）場合に使うと大変便利です。
- Starter では、プログラムコードとは別に出力結果を表示するための「`//output`」があります。プログラムコードと出力結果では見た目を変えて表示したい場合に便利です。
- Re:VIEW では、プログラムコードをハイライト表示する機能があります。
Starter ではまだ未サポートです。

A.10 ターミナル画面

- Starter では、ターミナル画面を表すコマンド命令「`//terminal`」を用意しています。これは使い方が「`//list`」と同じなので、覚えやすいです。
Re:VIEW で用意しているブロック命令「`//cmd`」は、「`//list`」と使い方が違うので覚えにくいです。
- Starter の「`//terminal`」はほぼ「`//list`」と同じなので、折返しや行番号など「`//list`」と同じ機能が「`//terminal`」でも使えます。
Re:VIEW の「`//cmd`」はそうではないので、機能が貧弱です。
- Starter では、ユーザ入力を表すインライン命令「`@<userinput>{}`」があります。これを使うと、ユーザ入力部分に薄い下線が引かれます。ただし $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ の制限により、長い行が右端で折り返しされません。
- Starter では、カーソルを表すインライン命令「`@<cursor>{}`」が用意されています。Vim の操作画面を説明するときに便利です。

A.11 ノート、ミニブロック

- Starter では、ノートを表すブロック命令「`//note`」を上書きし、見栄えのいい表示にしています。

Re:VIEW ではノートの見栄えは気にされていません。

- Starter では、ノートの中に箇条書きや他のブロック命令を入れられます。
- Starter では、ノートの中で脚注が使えます。
- Starter では、ノートの中に書いたプログラムコードやターミナル画面が、ページまたぎできます。
- Starter では、ノートにラベルをつけてあとから参照できます。これはちょうど、図や表を参照するのと同じような機能です。
- Starter では、補足情報を表す「`//info`」と注意喚起を表す「`//caution`」と警告を表す「`//warning`」にアイコンがつきます。

A.12 図

- Re:VIEW では、たとえば「`//image[]][scale=0.5]`」のようなオプションを指定すると、画像が本文幅の半分の大きさで表示されます。しかしこれは画像の幅が本文幅より大きい場合だけであり、画像の幅が本文幅より小さい場合は画像幅の半分の大きさで表示されます。そのため、画像の幅が本文幅より大きいかどうかを気にする必要があり、面倒です。

Starter では「`//image[]][width=50%]`」のように指定すると、画像の幅が本文幅より大きいかどうかに関係なく、常に本文幅の半分の大きさで画像が表示されます。

- Starter では「`//image[]][width=40mm]`」のように、画像の表示幅を絶対値で指定できます。つまり画像の表示幅を本文の幅とは関係なく指定できます。
- 画像の幅が本文幅より小さい場合、表示幅を指定せずに画像を表示すると、Re:VIEW では PDF ならそのままの大きさで表示されるのに、HTML では本文幅いっぱいの大きさで表示されてしまいます。

Starter では、PDF と HTML のどちらでもそのままの大きさで画像が表示されます。

- Starter では、画像の挿入位置を指定できます（現在位置、ページ上部、ページ下部、別ページ）。
- Starter では、図のまわりをグレーの線で囲む機能があります。画像のふちが白い場合はこの機能を使うと、画像の境界を明示できます。
- 画像が次のページに送られたとき、Starter では後続のテキストを現在位置に流し込まれます。

Re:VIEW では後続のテキストが現在位置に流し込まれないので、現在位置に大きなスペースが空きます。

- 「`//indepimage`」の第 2 引数に説明文字列を指定した場合、Re:VIEW では「図：」という接頭辞がつきます。しかし「`//indepimage`」の使用用途を考えたとき、この接頭辞は邪魔なので、Starter ではつけません。もし必要なら自力で「図：」を入れてください。
- Starter では、「`//sideimage`」を使うと画像の横にテキストを表示できます。

A.13 表

- Starter では、表の挿入位置を指定できます（現在位置、ページ上部、ページ下部、別ページ）。
- Starter では、表のデータを CSV 形式で指定できます。
- Starter では、表のデータを外部ファイルから読み込めます。
- Starter では、行ごとの罫線をオン・オフできます。また行ごとの罫線をオフにしたうえで、指定箇所だけ罫線を引くよう指定できます。
- Starter では、「`//table[label][]`」のようにラベルだけ指定してキャプションがない場合でも、「表 1.1」のような番号がつきます。

Re:VIEW では、ラベルが指定されてあってもキャプションがなければ番号が表示されません。

- Starter では、「`//table[?][]`」のように指定すると内部でラベルを自動的に割り当ててくれます。そのおかげで、表を番号つきで表示するのが簡単です。

Re:VIEW ではこのような機能がないので、番号つきで表示するには重複しないラベルを必ず指定する必要がある、面倒です。

- 表の列の右寄せや左寄せを指定する「`//tsize`」を、Starter では「`//tsize[latex][lcr]`」のように指定できます。

Re:VIEW では「`//tsize[|latex||lcr|]`」のように指定するため、読みづらいし分かりにくいです。

- Re:VIEW には、項目の区切り文字を変更できる機能があります。

Starter では未サポートです。

A.14 会話形式

- Starter では、アイコン画像付きの会話形式を用意しています。
- Starter では、アイコン画像なしの会話形式も用意しています。
- Starter では、大量の会話形式を入力するときに便利のように、短縮名を登録して短く書ける機能を用意しています。

A.15 用語、索引

- Starter には、用語を表す「@<term>{ }」インライン命令があります。これは「@<idx>{ }」と似ていますが、用語をゴシック体で表示する点が違います。用語が初めて登場したときは「@<term>{ }」、再登場した場合は「@<idx>{ }」を使うことを想定しています。
- Starter では、「@<idx>{ }」や「@<term>{ }」の中によみがなを指定できます。
- Starter では、親子関係と表示順序が逆転している場合にも「~~~」を使えば対応できます。
- Starter では、索引ページにおいて用語の転送先となる別の用語を指定できます。
- Re:VIEW の索引ページは、 \LaTeX のデフォルトデザインのままです。Starter では、用語とページ番号の間を連続したドットで埋めたり、用語のグループ化を文字単位から行単位に変更しています。

A.16 表紙、大扉、奥付

- Starter では、表紙と裏表紙は PDF ファイルで指定します。

Re:VIEW では、表紙は画像ファイル（PNG や JPG）で指定します。

- 本のタイトルが長い場合、Starter ではタイトルを複数行で指定すれば、改行位置を指定できます。このおかげで、大扉では長いタイトルがきれいに改行されて表示されます。

Re:VIEW ではこのような機能はありません。

- Starter では、大扉の裏に免責事項などが書かれています。これは商業誌では一般的ですが、同人誌では入れ忘れることがほとんどなので、あらかじめ Starter が用意しています。
- Starter では、奥付が必ず偶数ページ（見開きで左側のページ）になるよう調整します。

Re:VIEW ではこのような機能はありません^{*1}。

^{*1} TechBooster のテンプレートでは、Starter と同じように奥付が偶数ページになります。

A.17 PDF ファイル

- Starter では、印刷用 PDF では白黒に、電子用 PDF ではカラーにしてくれます。

Re:VIEW ではこのような機能はありません。

- Starter では、印刷用 PDF では左右の余白幅を切り替えています。これは本文幅を広げるための工夫であり、商業誌でも一般的です。

Re:VIEW ではこのような工夫はされていません。

- Re:VIEW では、印刷用 PDF にはハイパーリンクがつかないようにになっています。これは入稿時のトラブルをできるだけ減らすための仕様です。

しかし L^AT_EX ではハイパーリンクのある・なしで組版結果が変わることがあります (hyperref.sty の仕様)。また最近の印刷所では、ハイパーリンクがついた PDF でも入稿でトラブルになることはなさそうです。そこで Starter では、印刷用 PDF でもハイパーリンクを残したままにしています。

- Re:VIEW では、デフォルトで PDF にトンボがつきます。しかしこれは入稿に慣れてないとトラブルのもとです。入稿先が同人誌向けの印刷会社なら、トンボなしのほうが入稿トラブルは少ないです。

そのため、Starter ではトンボはつけません。

- Re:VIEW と Starter のどちらとも、印刷用 PDF には隠しノンブル（通し番号）がつきます。ただし実装方法は Re:VIEW と Starter とで全く別です。また Starter では、ノンブルの位置や大きさや色や開始番号を設定ファイルで変更できます。

- 印刷用 PDF に自動でノンブルがつくのは、PDF の作成が Re:VIEW だけで完結している場合は便利ですが、たとえば大扉を別ソフトで作るような場合は逆に不便です。

そこで Starter では、PDF を生成したあとに隠しノンブルをつける Rake タスク「`rake pdf:nombre`」も用意しています。これだと、大扉だけを別ソフトで作るような場合にも対応できます。

- Starter では、PDF ファイルにノンブルをつける別のサービス^{*2}も用意しています。
- Starter では、PDF のテキストをマウスで選択したときに、行番号や折り返し記号やインデント記号を選択対象にしない機能が用意されています。ただしこの機能はコンパイル時間を大きく増やしてしまうので、デフォルトではオフです。

^{*2} <https://kauplan.org/pdfoperation/>

A.18 L^AT_EX 関連

- Re:VIEW では、LuaLaTeX と jlreq.cls を積極的に利用しており、採用実績も多いです。

Starter では、今のところ LuaLaTeX と jlreq には対応していません。今後の課題です。

- Re:VIEW そのものではありませんが、Techbooster の Re:VIEW テンプレートでは tcolorbox を全面的に採用しており、そのおかげでたとえば角丸のブロックを実現できています。

ただし tcolorbox を使うとコンパイル時間がかかるようになるので、Starter では使わないようにしています。tcolorbox を使うと見栄えのいいデザインができることは確かなので、今後どうするかは検討中です。

- Starter では、環境変数「`$STARTER_CHAPTER`」を設定することで、特定の章だけをコンパイルできます。こうすると全部の章をコンパイルするよりずっと短い時間でコンパイルできるので、表示の確認が迅速にできます。

Re:VIEW にはこのような機能はありません。

- Starter では、L^AT_EX のコンパイル回数を減らす工夫をしています。文章を少しだけ変更して再度コンパイルすると、Starter ではコンパイル回数がたいてい 1 回だけで済みます。

Re:VIEW ではいつも 3 回コンパイルされるので、PDF 生成に時間がかかります。

- Re:VIEW では、PDF ファイルを生成する「`review-pdfmaker`」コマンドや、ePub ファイルを生成する「`review-epubmaker`」コマンドが用意されています。

Re:VIEW ではこれらは使えません。必ず「`rake pdf`」や「`rake epub`」を使ってください。

- 「`rake pdf`」や「`rake web`」を実行すると、Starter ではコンパイルが必ず実行されます。

Re:VIEW ではファイルが更新されていなければコンパイルされません。これは一見便利そうですが、L^AT_EX のスタイルファイルを変更しただけではコンパイルされないため、トラブルのもとになります。

- Starter では、設定ファイル「`config-starter.yml`」の内容を L^AT_EX のスタイルファイルから参照できます。設定によって挙動を変えたい場合に便利です。
- Starter では、名前が「`STARTER_`」で始まる環境変数の値を L^AT_EX のスタイルファイルから参照できます。環境変数によって挙動を変えたい場合に便利です。
- Starter では表示を簡潔にするため、L^AT_EX コンパイル時の出力を捨てています。コンパイルエラーがあった場合は、出力を捨てないようにコマンドオプションを変更してからもう一度コンパイルすることで、エラーメッセージを表示しています。

Re:VIEW でも出力は捨てていますが、コンパイルエラーがあった場合はコンパイル時のログファイルを表示しています。しかしこの方法だとエラーメッセージが少し上のほうに流れてしまうので、L^AT_EX に慣れていない人だとエラーメッセージを見逃してしまいます。

- PDF ファイルを生成するとき、Starter ではもとの PDF ファイルを削除せずに上書きする（つまり inode が変わらない）ようにしています。このおかげで、macOS の `open` コマンド

で PDF ファイルを開くと、同じウィンドウのまま表示されます。

これに対し、Re:VIEW ではもとの PDF ファイルを消してから新しいファイルを作成します（つまり inode が変わる）。そのせいで、macOS の `open` コマンドで PDF ファイルを開くと毎回新しいウィンドウで表示されてしまい、いちいち前のウィンドウを閉じる必要があるので面倒です。

- Starter では、 \LaTeX のコンパイル時間を表示します。以前は「`/usr/bin/time`」コマンドが必要でしたが、現在は必要としません。

A.19 エラメッセージ

- Starter では、原稿ファイル (*.re) コンパイル時のエラーメッセージを赤い字で表示します。初心者はエラーメッセージが出ても気づかないことが多いので、赤く表示することで気づきやすいようにしました。
- Starter では、原稿ファイル (*.re) のコンパイル時にエラーがあると、すぐにコンパイルをすぐに終了するようにしました。初心者にとっては、このほうがエラーに気づきやすいはずです。

Re:VIEW ではコンパイルエラーがあっても残りのコンパイルを続けるので、エラーメッセージが流れてしまい、初心者はエラーに気づかない可能性が高いです。

A.20 その他

- Re:VIEW では、プロジェクトが使っている Re:VIEW のバージョンをアップデートする機能が用意されています。たとえば Re:VIEW 4 を使っているプロジェクトを Re:VIEW 5 へアップデートしたいときは、ターミナルで「**review-update**」コマンドを実行します。
Starter にはプロジェクトをアップデートする機能はありません。
- Starter では関連プロジェクトとして、PDF に隠しノンブルをつけるサービス「PDF Operation^{*3}」を提供しています。

^{*3} <https://kauplan.org/pdfoperation/>

付録 B

ファイルとフォルダ

プロジェクトの zip ファイル（例：mybook.zip）を解凍してできるファイルとフォルダの解説です。

README.md

プロジェクトの説明が書かれたファイルです。ユーザが好きなように上書きすることを想定しています。

Rakefile

rake コマンドが使用します。コマンドを追加する場合は、このファイルは変更せず、かわりに「lib/tasks/*.rake」を変更してください。

catalog.yml

原稿ファイルが順番に並べられたファイルです。原稿ファイルを追加した・削除した場合は、このファイルも編集します。

config-starter.yml

Starter 独自の設定ファイルです。Starter では設定ファイルとして「cofnig.yml」と「cofnig-starter.yml」の両方を使います。

config.yml

Re:VIEW の設定ファイルです。Starter によりいくつか変更と拡張がされています。

contents/

原稿ファイルを置くフォルダです*¹。

contents/*.re

原稿ファイルです。章 (Chapter) ごとにファイルが分かります。

css/

HTML ファイルで使う CSS ファイルを置くフォルダです。ePub で使うのはこれではなくて style.css なので注意してください。

images/

画像ファイルを置くフォルダです。この下に章 (Chapter) ごとのサブフォルダを作ることできます。

*¹ 原稿ファイルを置くフォルダ名は「config.yml」の「contentdir: contents」で変更できます。

layouts/layout.epub.erb

原稿ファイルから ePub ファイルを生成するためのテンプレートです。

layouts/layout.html5.erb

原稿ファイルから HTML ファイルを生成するためのテンプレートです。

layouts/layout.tex.erb

原稿ファイルから LaTeX ファイルを生成するためのテンプレートです。

lib/hooks/beforetexcompile.rb

LaTeX ファイルをコンパイルする前に編集するスクリプトです。

lib/ruby/*.rb

Starter による Re:VIEW の拡張を行う Ruby スクリプトです。

lib/ruby/mytasks.rake

ユーザ独自の Rake コマンドを追加するためのファイルです。

lib/ruby/review.rake

Re:VIEW で用意されている Rake タスクのファイルです。Starter によって変更や拡張がされています。

lib/ruby/review.rake.orig

Starter によって変更や拡張がされる前の、オリジナルのタスクファイルです。

lib/ruby/starter.rake

Starter が追加した Rake タスクが定義されたファイルです。

locale.yml

国際化用のファイルです。たとえば「リスト 1.1」を「プログラム 1.1」に変更したい場合は、このファイルを変更します。

mybook-epub/

ePub ファイルを生成するときの中間生成ファイルが置かれるフォルダです。通常は気にする必要はありません。

mybook-pdf/

PDF ファイルを生成するときの中間生成ファイルが置かれるフォルダです。LaTeX ファイルをデバッグするときが必要となりますが、通常は気にする必要はありません。

mybook.epub

生成された ePub ファイルです。ファイル名はプロジェクトによって異なります。

mybook.pdf

生成された PDF ファイルです。ファイル名はプロジェクトによって異なります。

review-ext.rb

Re:VIEW を拡張するためのファイルです。このファイルから「lib/ruby/*.rb」が読み込まれています。

sty/

LaTeX で使うスタイルファイルが置かれるフォルダです。

sty/jumoline.sty

L^AT_EX で使うスタイルファイルのひとつです。

sty/mycolophon.sty

奥付^{*2}の内容が書かれたスタイルファイルです。奥付を変更したい場合はこのファイルを編集します。

sty/mystyle.sty

ユーザが独自に L^AT_EX マクロを定義・上書きするためのファイルです。中身は空であり、ユーザが自由に追加して構いません。

sty/mytextsize.sty

PDF における本文の高さと幅を定義したファイルです。L^AT_EX では最初に本文の高さと幅を決める必要があるので、他のスタイルファイルから分離されてコンパイルの最初に読み込めるようになっています。

sty/mytitlepage.sty

大扉^{*3}の内容が書かれたスタイルファイルです。大扉のデザインを変更したい場合はこのファイルを編集します。

sty/starter.sty

Starter 独自のスタイルファイルです。ここに書かれた L^AT_EX マクロを変更したい場合は、このファイルを変更するよりも「sty/mystyle.sty」に書いたほうがバージョンアップがしやすくなります。

sty/starter-codeblock.sty

プログラムコードやターミナルの L^AT_EX マクロが定義されています。

sty/starter-color.sty

色のカスタマイズ用です。

sty/starter-font.sty

フォントのカスタマイズ用です。

sty/starter-headline.sty

章 (Chapter) や節 (Section) や項 (Subsection) の L^AT_EX マクロが定義されたファイルです。

sty/starter-note.sty

ノートブロックの L^AT_EX マクロが定義されています。

sty/starter-section.sty

以前の、章や節の L^AT_EX マクロ定義です。もはや使ってませんが、starter.sty を書き換えれば使えます。

sty/starter-talklist.sty

会話形式の L^AT_EX マクロが定義されています。

^{*2} 奥付とは、本のタイトルや著者や出版社や版や刷などの情報が書かれたページのことです。通常は本のいちばん最後のページに置かれます。

^{*3} 大扉とは、タイトルページのことです。表紙のことではありません。

sty/starter-toc.sty

目次のカスタマイズ用です。

sty/starter-misc.sty

その他、各種のマクロ定義です。

sty/starter-util.sty

Starter のスタイルファイルで使われるマクロが定義されています。

style.css

ePub で使われる CSS スタイルファイルです。

付録 C

開発の背景

ここでは歴史の記録として、Re:VIEW Starter が開発された背景などを記しておきます。個人的な忘備録として残すものであり、読まなくても本や同人誌の制作にはまったく支障はありません。

開発の経緯

Re:VIEW は ver 2.4 の頃、「A5 サイズの PDF ファイルが生成できない」「フォントサイズも 10pt から変更できない」という致命的な問題を抱えていました。つまり B5 サイズでフォントが 10pt の本しか作れなかったのです。

この頃はまだ、技術同人誌は B5 サイズ 10pt で作るのが主流だったので、Re:VIEW のこの制限はあまり問題にはなりませんでした。しかし同人誌印刷では B5 サイズより A5 サイズのほうが割安なので、一部の人が A5 サイズやより小さいフォントサイズでの PDF 生成を模索し、そして Re:VIEW の制限に苦しみました。

たとえば、フォントサイズを小さくしようとして格闘し、結果としてあきらめた人の証言を見てみましょう*¹ (図 C.1)。

この問題は、「**geometry.sty**」というスタイルファイルをオプションなしで読み込んでいることが原因です*²。具体的な修正方法は Qiita の記事*³に書いています。読めば分かりますが、かなり面倒です。

この不具合はバグ報告したもの、開発陣の反応は芳しくなく、すぐには修正されなさそうでした*⁴。当時は「技術書典」という同人イベントの開催が迫っていたので、これは困りました。仕方ないので Re:VIEW 側での修正に期待せず、誰でも簡単に A5 サイズの同人誌が作れるための別の方法を考えることにしました。

*¹ <https://www.slideshare.net/KazutoshiKashimoto/nagoya0927-release> の p.21 と p.22。

*² 簡単に書いてますが、原因が **geometry.sty** であることを突き止めるのには多くの時間がかかり、正月休みが潰れました。今でも恨めます。この苦勞を知らずに「Re:VIEW では昔から A5 の PDF が簡単に生成できた」と言い張る歴史修正者にはケツパットの刑を与えてやりたいくらいです。

*³ <https://qiita.com/kauplan/items/01dee0249802711d30a6>

*⁴ 実際、Re:VIEW Ver 2 の間は修正されず、修正されたのは Ver 3 になってからでした。当然、技術書典には間に合いませんでした。

引用： <https://www.slideshare.net/KazutoshiKashimoto/nagoya0927-release>

(p.21)

(p.22)



▲ 図 C.1: フォントやページサイズを変更できなかった人の証言

そうした経緯で誕生したのが、Re:VIEW Starter です。Starter では初期設定が GUI でできる簡単さが売りのひとつですが、もともとは A5 サイズ 9pt の同人誌を簡単に作れることが開発動機だったのです。

開発の転機

Starter は、当初は Re:VIEW との違いが大きくなならないよう、GUI で設定した状態のプロジェクトをダウンロードできるだけに留めていました。つまりユーザが手作業で設定するのを、GUI で簡単に設定できるようにしただけでした。

しかし Re:VIEW は、ドキュメント作成ツールとしての基本機能が足りておらず、そのうえ開発速度が遅くて技術書典といったイベントには新機能追加が間に合いません。またバグ報告をしても「仕様だ」と回答されたり、互換性を理由に却下されたり（けど他の人が報告すると取り入れられたり）といったことが積み重なって、ある日を境に Starter で独自の機能を追加することを決心しました。

そこからは少しずつ Re:VIEW の機能を上書きし、足りない機能を追加していきました。

現在の Starter では、Re:VIEW のソースコードを広範囲に上書きしています^{*5}。特にパーサの大部分は Starter 側で上書きしています。このおかげで、インライン命令やブロック命令を入れ子対応にしたり、箇条書きの機能を大きく拡張したりできています。

Re:VIEW とのかい離はずいぶんと大きくなりました。

^{*5} 実は Re:VIEW のソースコードを広範囲に上書きしているせいで、ベースとなる Re:VIEW のバージョンを 2.5 から上げられていません。しかし Re:VIEW の開発速度が遅く目ぼしい新機能は追加されていないので、デメリットはありません。

開発の障害

Starter を開発するうえで大きな障害になったのが、 \LaTeX と、Re:VIEW のコード品質です。

「 \LaTeX 」はフリーの組版ソフトウェアであり、Re:VIEW や Starter が PDF ファイルを生成するときに内部で使っています。 \LaTeX は出力結果がとてもきれいですが、使いこなすには相当な知識が必要です。

特に \LaTeX のデバッグは困難を極めます。エラーメッセージがろくに役立たないせいでエラーを見ても原因が分からない、どう修正すればいいかも分からない、直ったとしてもなぜ直ったのか分からない、分からないことだらけです。この文章を読んでいる人にアドバイスできることがあるとすれば、「 \LaTeX には関わるな！」です。

また Re:VIEW のコード品質の悪さも、大きな障害となりました。ソースコードを読んでも何を意図しているのか理解しづらい、コードの重複があちこちにある、パーサで行うべきことを Builder クラスで行っている、…など、基礎レベルでのリファクタリングがされていません。信じたくはないでしょうが、同じような感想を持った人が他にもいたので紹介します。

<https://np-complete.gitbook.io/c86-kancolle-api/atogaki>:

っと上の文章を書いたから 2 時間ほど Re:VIEW のコードと格闘していたんですが、なんですかこのクソコードは・・・これはマジでちょっとビビるレベルのクソコードですよ。夏コミが無事に終わったら冬に向けて Re:VIEW にプルリク送りまくるしかないと思いました。

「クソコード」は言い過ぎですが、そう言いたくなる気持ちはとてもよく分かります。

また、パースしたあとに構文木を作っていないのは、Re:VIEW の重大な設計ミスといえるでしょう。Re:VIEW や Markdown のようなドキュメント作成ツールでは、一般的に次のような設計にします。

1. パーサが入力テキストを解析して、構文木を作る。
2. 構文木をたどって必要な改変をする。
3. Visitor パターンを使って、構文木を HTML や \LaTeX のコードに変換する。

このような設計であれば、機能追加は容易です。特に HTML や \LaTeX のコードを生成するより前にすべての入力テキストがパース済みなので、たとえ入力テキストの最後に書かれたコマンドであろうと、それを認識して先頭のコードを柔軟に変更できます。

しかし Re:VIEW は構文木を作らず、パースしながら HTML や \LaTeX のコードに変換するため、しなくてもいいはずのハックが必要となることがあります。

Starter は将来的に Re:VIEW のパーサをすべて上書きして、構文木を生成するタイプのパーサへ置き換えることになるでしょう。

開発方針の違い

Re:VIEW と Starter では、開発方針に大きな違いがあります。

開発速度

Re:VIEW では、リリースが年 3 回と決まっています。またリリース時期も（過去を見る限り）2 月末、6 月末、10 月末に固定されています。そのため「次の技術書典までにこの新機能が必要だ」と思っても、固定されたリリース時期にならないと新機能はリリースされません。言い方を変えると、ユーザが必要とする開発速度についてこれていません。

Starter は主に、技術書典をはじめとした同人イベントに合わせて新機能が開発されます。また 2〜3 週間ごとにリリースされるので、新機能の追加とバグ修正が急ピッチで進みます。つまり、ユーザが必要とする速度で開発されています。

機能選定

Re:VIEW は商業誌での利用実績が多いこともあって、「日本語組版処理の要件^{*6}」への対応が重視されます。そのため `jlreq` クラスファイル^{*7}が標準でサポートされており、少なくない開発リソースがその対応に割かれています。そのせいか、「範囲コメントを実装する」「インライン命令やブロック命令を入れ子に対応させる」「順序つき箇条書きで数字以外を使えるようにする」などの基本機能が、未だに実装されていません。

しかし「日本語組版処理の要件」は、読んでみると分かりますが、重箱の隅をつつくような内容がほとんどです。プロユースでは必要なかもしれませんが、同人誌では「禁則処理がきちんとできていれば充分」というユーザがほとんどでしょう。開発リソースをそんな細かいところに割くよりも、もっとユーザが必要とする機能の開発に割くべきです。

Starter では日本語組版の細かい要件は気にせず、ユーザが必要とする機能を重点的に開発しています。たとえば、範囲コメントを実装したり、インライン命令やブロック命令を入れ子に対応させたり、順序つき箇条書きでアルファベットも使えるようにしたり、章や節のタイトルを見栄えのいいデザインにしたりといった、ユーザにとって必要な機能を優先して開発しています^{*8}。

バグ対応

今まで、Re:VIEW には約 20 個ほどのバグ報告や Pull Request を出しました。取り入れられたものも多いですが、理不尽な理由で拒絶されたものも多いです。

明らかなバグを仕様だと言い張る

たとえば、Re:VIEW では箇条書きの項目が複数行だと勝手に結合されるというバグがあります。

▼ サンプル

```
* AA AA
  BB BB
```

^{*6} <https://www.w3.org/TR/jlreq/ja/>

^{*7} <https://github.com/abenori/jlreq>

^{*8} ここに挙げた機能のうち、見た目のデザインを変更するのは (LaTeX の知識さえあれば) ユーザでも変更できますし、実際に Techbooster テンプレートでは見た目のデザインを変更しています。しかしそれ以外の機能は Re:VIEW のソースコードを変更しなければ実現できず、ユーザが簡単に行えることではありません。

▼ 表示結果

- AA AABBBBCC CC

「AABB」や「BBCC」のように英単語が結合されていますよね？ どう見ても Re:VIEW のバグなのですが、報告しても「これは仕様だ」と言い張るんです。しかも「なぜそのような書き方をするんだ？ 一行に書けばいいではないか」と言われる始末。かなり意味不明な対応をされました。

Starter ではこのバグは修正しています（当然です）。

提案を却下しておきながら次のバージョンで釈明なく取り入れる

Re:VIEW 2.x では、 \LaTeX スタイルファイルが「sty/reviewmacro.sty」しかなく、カスタマイズするにはこのファイルを編集するのが一般的でした。しかしこの方法には問題があります。

- 「sty/reviewmacro.sty」を直接編集すると、Re:VIEW のバージョンアップをするときに困る。
- 自分でスタイルファイルを追加できるが、そのためには設定ファイルの項目を変更する必要があり、初心者には敷居が高い（どの項目をどう変更すればいいか知らないため）。

そこで、あらかじめ空のスタイルファイルを用意し、ユーザのカスタマイズはそのファイルの中で行うことを提案しました。Starter でいうと「sty/mystyle.sty」のことですね。実装は簡単だし、問題の解決策としてとても妥当な方法です。

しかしこの提案は、あーだこーだと理由をつけられて却下されました（これに限らず、初心者への敷居を下げるための提案は却下されることが多い印象です）。仕方ないので、Starter では初期の頃から独自に空のスタイルファイルを提供していました。

ところが、なんと Re:VIEW 3.0 でこの機能が取り込まれていたのです！ 「提案が通ったならそれでいいじゃないか」と心ないことを言う人もいるでしょうけど、いろいろ理由をつけて提案を却下しておきながら、何の釈明もなくしれーっと取り込むのは、却下されたほうとしてはたまったものではありません。不満が残るのは当たり前です。

仲のいい人とそうでない人とで露骨に態度を変える

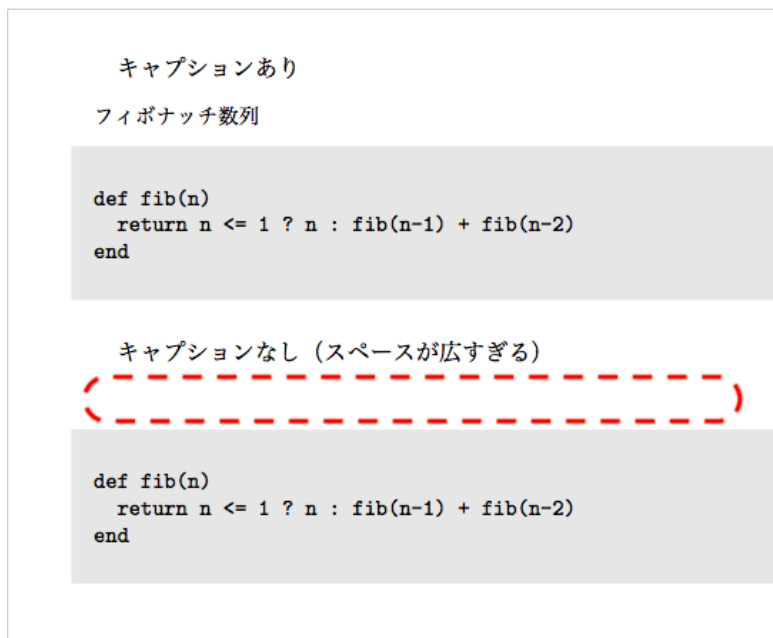
昔の Re:VIEW では、「//list」においてキャプション（説明文）を指定しなくても言語指定をすると、本文とプログラムリストの間が大きく空いてしまうというバグがありました。たとえばこのように書くと：

▼ サンプル

```
//list[[]][ruby]{
def fib(n)
  return n <= 1 ? n : fib(n-1) : fib(n-2)
end
```

```
//}
```

図 C.2 の下のように表示されていました。



▲ 図 C.2: 本文とプログラムリストの間が大きく空いてしまう (下)

この現象は、空文字列がキャプションとして使われるため、その分の空行が空いてしまうことが原因です。そこで、キャプションが空文字列のときは表示しないようにする修正を報告しました。

しかしこのバグ報告も、後方互換性のために却下されました。「出力結果が変わってしまうような変更は、たとえバグ修正だとしても、メジャーバージョンアップ以外では受けつけられない」というのが理由でした。こんな明白なバグの互換性なんかいらんはずだと思ったので、粘って交渉したのですが、互換性を理由に頑なに拒まれました。

ところが、開発者と仲のいい別の人が「やはりバグではないか？」とコメントした途端、メジャーバージョンアップでもないのに修正されました。見事な手のひら返しです。

「出力結果が変わってしまう変更は、たとえバグ修正でも（メジャーバージョンアップ以外では）受けつけられない。それが組版ソフトだ」という理由でさんざん拒絶しておいて、いざ別の人が言及するとすぐに修正する。出力結果が変わってしまう修正だというのに！

大事な点なので強調しますが、**出力結果が変わる変更はダメと言って拒絶しておきながら、別の人が「やはりバグでは？」と言うと出力結果が変わる変更でもあっさり行うという、露骨な態度の違い。**よく平気でこんなことできるなど逆に感心しました。

こんなことが積み重なったので、もうバグ報告も機能提案もしないことにしました。Re:VIEWのコードは品質が良くないので、ソースコードを読んでいると細かいバグがちょこちょこ見つかり

ますが、もうシラネ。

また Re:VIEW のリリースノートを見ると、Starter の機能が Re:VIEW の新機能として取り込まれているのを見かけます。つまり、**Re:VIEW に新機能提案しても通らないけど、Starter に機能を実装すると Re:VIEW に取り込まれる可能性が高い**ということです。こちらはバグ報告や提案が変な理由で却下されてストレスが溜まることから解放されるし、Re:VIEW 側は Starter が先行実装した機能を選んで取り込めばいいし、Win-Win で、これでいいのだ。

今後の開発

- Starter は、将来的に Re:VIEW のソースコードをすべて上書きするでしょう。**大事なのはユーザが書いた原稿であって、Re:VIEW でも Starter でもありません。**たとえ Re:VIEW のソースコードをすべて捨てたとしても、ユーザの原稿は使えるようになるはずです。
- 設定は、今は設定ファイルを手作業で変更していますが、これは GUI による設定に置き換わるでしょう。そのため設定ファイルの互換性はなくなる可能性が高いです。
- PDF 生成のための組版ツールとして、 \LaTeX 以外に Vivliostyle をサポートしたいと考えています。ただし Vivliostyle プロジェクトが独自のドキュメント作成ツールを開発中なので、Vivliostyle を必要とする人はそちらを使うだろうから、Starter での優先順位は高くなくてもいいかなと思っています。
- 同人イベントがオンラインに移行することから、PDF より ePub の需要が高くなるでしょう。今の Starter は ePub 対応が弱いので、強化する必要があります。
- 印刷物での配布が減るので、カラー化がいつそう進むでしょう。Starter はコードハイライトができないので、対応を急ぐ必要があります。
- Visual Studio Code での執筆を支援するプラグインが必要とされるでしょう。そのための本を買ったけど、積ん読のままです。
- Markdown ファイルの対応は、パーサを書き換えたあとで検討します。
- 諸般の事情により Git リポジトリを公開していないので、公開できるよう準備を進めます。

あとがき / おわりに

いかがだったでしょうか。感想や質問は随時受けつけています。

著者紹介



カウプラン機関極東支部 (@_kauplan)

“賞味期限が1年にも満たないようなツールやフレームワークに振り回されるのを許しておけるほど、我々の人生は長くはない。”

- <https://kauplan.org/>
- 『パンプキン・シザーズ』 推し
- 『ワールド・トリガー』 推し
- 『プリンセス・プリンシパル』 推し

既刊一覧

- 『SQL 高速化 in PostgreSQL』 (技術書典 2)
- 『オブジェクト指向言語解体新書』 (技術書典 3)
- 『jQuery だって複雑なアプリ作れるもん!』 (技術書典 4)
- 『Shell スクリプトでサーバ設定を自動化する本』 (技術書典 5)
- 『Ruby のエラーメッセージが読み解けるようになる本』 (技術書典 6)
- 『わかりみ SQL』 (技術書典 7)
- 『Python の黒魔術』 (技術書典 8)
- 『カウプランレポート vol.01』 (技術書典 10)

課長のせいで今日も死んでます。楽しんで覚える情報技術

2025 年 5 月 30 日 ver 1.0 (技術書典 18)

著 者 株式会社おもしろテクノロジー
印刷所 日光企画

© 2025 株式会社おもしろテクノロジー

(powered by Re:VIEW Starter)