Re:VIEW Starter ユーザーズガイド

— インストールからカスタマイズまで —

カウプラン機関極東支部　著

2020 年 7 月 7 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、™、Ⓡ、Ⓒ などのマークは省略しています。

まえがき

＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊ **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_123›
›456789_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-narrow）

```
123456789_123456789_123456789_123456789_123456789_123456789_123456›
›789_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
八九十一二三四五六七八九十
```

＊＊＊＊＊＊＊＊＊＊＊ 以下、まえがきのサンプル ＊＊＊＊＊＊＊＊＊＊＊

本書を手に取っていただき、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。

目次

まえがき i

第 1 章 インストール 1

1.1 Ruby と TeXLive のインストール . . . 2
- ♣ Docker の使い方を知っている場合 . . . 2
- ♣ 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 . . . 5

1.3 サンプルの PDF ファイルを生成 . . . 5
- ♣ Docker の場合 . . . 6
- ♣ macOS の場合 . . . 7
- ♣ Windows の場合 . . . 7

1.4 注意点 . . . 8

第 2 章 チュートリアル 9

2.1 用語の説明 . . . 10

2.2 フォルダとファイルの説明 . . . 11

2.3 原稿の追加と変更 . . . 12
- ♣ 既存の原稿ファイルを変更する . . . 12
- ♣ 新しい原稿ファイルを追加する . . . 14
- ♣ 「catalog.yml」の説明 . . . 16
- ♣ コンパイルエラーになったら . . . 17

2.4 基本的な記法 . . . 17
- ♣ コメント . . . 18
- ♣ 段落と改行 . . . 18
- ♣ 見出し . . . 19
- ♣ 箇条書き . . . 19
- ♣ 用語リスト . . . 20
- ♣ 太字と強調 . . . 21
- ♣ プログラムリスト . . . 22
- ♣ ターミナル画面 . . . 23
- ♣ 脚注 . . . 24
- ♣ 図 . . . 24
- ♣ ノート . . . 25

♣ 表 . . . 26
♣ 数式 . . . 26
2.5 印刷用 PDF と電子用 PDF . . . 27
♣ 印刷用と電子用を切り替えて PDF を生成する . . . 27
♣ 印刷用 PDF と電子用 PDF の違い . . . 28
♣ ノンブル . . . 28
2.6 高速プレビュー . . . 29
♣ 指定した章だけをコンパイルする . . . 30
♣ 画像読み込みを省略するドラフトモード . . . 30
♣ 自動リロードつき HTML プレビュー . . . 31

第 3 章 記法 33
3.1 コメント . . . 34
♣ 行コメント . . . 34
♣ 範囲コメント . . . 34
3.2 段落と改行と空行 . . . 35
♣ 段落 . . . 35
♣ 改行 . . . 36
♣ 強制改行 . . . 37
♣ 強制空行 . . . 38
3.3 見出し . . . 38
♣ 見出しレベル . . . 38
♣ 見出し用オプション . . . 39
♣ 章の概要 . . . 40
♣ 章 ID . . . 40
♣ 章を参照する . . . 41
♣ 節や項を参照する . . . 41
♣ まえがき、あとがき、付録 . . . 43
♣ 部 . . . 43
♣ 段見出しと小段見出し . . . 44
3.4 箇条書き . . . 47
♣ 番号なし箇条書き . . . 47
♣ 番号つき箇条書き . . . 47
3.5 用語リスト . . . 48
3.6 インライン命令とブロック命令 . . . 49
♣ インライン命令 . . . 49
♣ ブロック命令 . . . 50
3.7 強調と装飾 . . . 51

♣ 強調 . . . 51
♣ 太字 . . . 52
♣ 装飾 . . . 52
♣ 文字サイズ . . . 52
♣ マーキング用 . . . 52
3.8 プログラムリスト . . . 53
♣ 基本的な書き方 . . . 53
♣ リスト番号 . . . 54
♣ ラベルの自動指定 . . . 55
♣ 長い行の折り返し . . . 55
♣ 改行文字を表示 . . . 57
♣ インデント . . . 58
♣ 外部ファイルの読み込み . . . 58
♣ タブ文字 . . . 59
♣ その他のオプション . . . 59
3.9 行番号 . . . 59
♣ プログラムリストに行番号をつける . . . 59
♣ 行番号の幅を指定する . . . 60
♣ 複雑な行番号指定 . . . 61
3.10 ターミナル . . . 62
3.11 脚注 . . . 64
3.12 ノート . . . 64
♣ ノートの書き方 . . . 64
♣ ラベルをつけて参照する . . . 66
♣ ノート以外のミニブロック . . . 67
3.13 コラム . . . 68
♣ コラムとは . . . 68
♣ コラムの書き方 . . . 68
[コラム] サンプルコラム 1 . . . 69
♣ コラム内の脚注 . . . 69
[コラム] サンプルコラム 2 . . . 70
♣ コラムを参照 . . . 70
[コラム] サンプルコラム 3 . . . 70
♣ コラムの制限 . . . 70
♣ ノートとコラムの使い分け . . . 71
3.14 画像 . . . 71
♣ 画像ファイルの読み込み方 . . . 71
♣ 章ごとの画像フォルダ . . . 72
♣ 画像のまわりに線をつける . . . 73

♣ 画像の配置位置を指定する 73
♣ 画像に番号も説明もつけない 75
♣ 文章の途中に画像を読み込む 75
♣ 画像とテキストを並べて表示する 75
3.15 表 77
♣ 表の書き方 77
♣ 右寄せ、左寄せ、中央揃え 77
♣ 罫線 78
♣ セル幅 79
♣ 空欄 79
3.16 数式 81
♣ 数式の書き方 81
♣ 数式を文中に埋め込む 81
♣ 数式のフォント 82
3.17 その他 82
♣ URL、リンク 82
♣ 引用 83
♣ 傍点 84
♣ ルビ、読みがな 84
♣ 何もしないコマンド 85
♣ コード中コメント 85
♣ 改ページ 86
♣ 特殊文字 86
♣ ターミナルのカーソル 86
♣ 右寄せ、センタリング 87
♣ 生データ 87
♣ 索引 88
♣ 参考文献 88
♣ 単語展開 88

第 4 章 カスタマイズ 89
4.1 命令 90
♣ 新しいインライン命令を追加する 90
♣ 新しいブロック命令を追加する 90
4.2 ページと本文 93
♣ [PDF] ページサイズを B5 や A5 に変更する 93
♣ [PDF] 本文の幅を指定する 93
♣ [PDF] 本文の高さを指定する 94
♣ [PDF] 章の右ページ始まりをやめる 94

♣ [PDF] 「//info」や「//warning」のデザインを変更する 95
4.3 フォント 96
♣ [PDF] フォントサイズを変更する 96
♣ [PDF] フォントの種類を変更する 96
4.4 プログラムコード 97
♣ [PDF] 説明文直後での改ページを防ぐ 97
♣ [PDF] プログラムコードのフォントを変更する 98
♣ プログラムコードの枠線 99
4.5 見出し 99
♣ 章のタイトルデザインを変更する 99
♣ 節のタイトルデザインを変更する 100
♣ 段見出しのデザインを変更する 100
♣ [PDF] 章や節のタイトルを独自にデザインする 100
♣ [PDF] 章扉をつける 100
♣ [PDF] 節ごとに改ページする 101
♣ 項に番号をつける 102
♣ 項を目次に含める 102
♣ [PDF] 項への参照に節タイトルを含める 102
♣ 項タイトルの記号を外す 103
♣ [PDF] 項タイトルの記号をクローバーから変更する 103
4.6 本 104
♣ 本のタイトルや著者名を変更する 104
♣ [PDF] 表紙を指定する 104
♣ [PDF] 大扉のデザインを変更する 105
♣ [PDF] 注意書きの文章を変更する 105
♣ [PDF] 奥付のデザインを変更する 105
4.7 Rake タスク 106
♣ デフォルトタスクを設定する 106
♣ 独自のタスクを追加する 106
♣ コンパイルの前処理を追加する 106

第 5 章 FAQ 109

5.1 コンパイルエラー 110
♣ コンパイルエラーが出たとき 110
♣ 「review-pdfmaker」で PDF が生成できない 111
5.2 本文 111
♣ 左右の余白が違う 111
♣ 文章中のプログラムコードに背景色をつけたい 111
♣ 索引をつけたい 112

5.3 プログラムコードとターミナル 112
♣ プログラムコードの見た目が崩れる 112
♣ 右端に到達してないのに折り返しされる 112
♣ 半角文字の幅を全角文字の半分にしたい 113
♣ ターミナルの出力を折り返しせずに表示したい 113
♣ コードハイライトしたい 115
5.4 その他 115
♣ LaTeX のスタイルファイルから環境変数を参照する 115
♣ 高度なカスタマイズ 115

第 6 章 よりよい本づくり 119
6.1 まえがき 120
♣ まえがきには章番号をつけない 120
♣ まえがきに「本書の目的」を入れる 120
♣ まえがきに「対象読者」を入れる 120
♣ まえがきに「前提知識」を入れる 121
♣ まえがきにサポートサイトの URL を載せる 121
♣ まえがきに謝辞を載せる 122
♣ まえがきにソフトウェアのバージョンを載せる 122
♣ まえがきの章タイトル以外は目次に載せない 122
6.2 初心者向け入門書 123
♣ 初心者向け入門書ではフォントを大きめにする 123
♣ 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 124
♣ 初心者向け入門書では節ごとに改ページする 124
6.3 箇条書き 125
♣ 箇条書きを正しく使い分ける 125
♣ 箇条書き直後に継続する段落は字下げしない 126
6.4 本文 128
♣ 強調箇所は太字のゴシック体にする 128
♣ 強調と傍点を使い分ける 128
♣ [PDF] 記号と日本語はくっつくことに注意する 129
♣ [PDF] 等幅フォントで余計な空白が入るのを防ぐ 130
♣ 文章中のコードは折り返しする 130
♣ ノートとコラムを使い分ける 130
6.5 見出し 130
♣ 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 130
♣ 項を参照するなら項番号をつける 131
6.6 プログラムコード 131
♣ 0 と O や 1 と l が見分けやすいフォントを使う 131

♣ フォントを小さくしすぎない 132
♣ 重要箇所を目立たせる 132
♣ 重要でない箇所を目立たせない 133
♣ 差分（追加と削除）の箇所を明示する 133
♣ 長い行の折り返し箇所を明示する 135
♣ 行番号を控えめに表示する 136
♣ 行番号を考慮して長い行を折り返す 136
♣ ページまたぎしていることを可視化する 136
♣ インデントを可視化する 137
6.7 図表 138
♣ 図表が次のページに送られてもスペースを空けない 138
♣ 大きい図表は独立したページに表示する 138
♣ 図表は番号で参照する 138
♣ 表のカラム幅を指定する 138
♣ 表のカラムが数値なら右寄せにする 139
6.8 ページデザイン 140
♣ [PDF] 印刷用では左右の余白幅を充分にとる 140
♣ [PDF] タブレット向けでは余白を切り詰める 142
♣ 電子書籍ではページ番号の位置を揃える 142
♣ 非 Retina 向けには本文をゴシック体にする 142
6.9 大扉 142
♣ 長いタイトルでは改行箇所を明示する 142
♣ 大扉を別のソフトで作成する 142
6.10 奥付 143
♣ 奥付は最後のページに置く 144
♣ 奥付に更新履歴とイベント名を記載する 144
♣ 利用した素材の作者と URL を奥付に記載する 144
♣ 利用したソフトウェアを奥付に記載する 144

付録 A Re:VIEW との差分 145

付録 B ファイルとフォルダ 147

付録 C 開発の背景 151

あとがき 161

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。

【この章の内容】

1.1 Ruby と TeXLive のインストール 2
1.2 プロジェクトを作成 5
1.3 サンプルの PDF ファイルを生成 5
1.4 注意点 8

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/Debi›
›an)
...(省略)...
```

♣ 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

[*1] Docker についてのこの説明は、技術的にはち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/

[*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 ＞

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

```
$ 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）
```

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たら、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）。

「ターミナル」をダブルクリックすると起動できます。

▼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 2.6.6-1 (x64)」をダウンロードしてインストールしてください。

- 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 2.6.6-1 [x64-mingw32]
```

実際の出力結果は上と少し違うと思いますが、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
```

[*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。

```
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。

▼ 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/W32T›
›eX)
...（以下省略）...
```

実際の出力結果は上と少し違うと思いますが、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「#reviewstarter」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「#reviewstarter」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。

[*5] 繰り返しになりますが、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。

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
```

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「#reviewstarter」タグをつけて質問してください（相手先不要）。

1.4　注意点

注意点をいくつか説明します。

- エラーが発生したときの対処法は「♣ コンパイルエラーになったら」(p.17) で簡単に説明しています。必ず読んでください。
- ePub ファイルを生成するときは「rake epub」または「rake docker:epub」、HTML ファイルを生成するときは「rake web」または「rake docker:epub」を実行してください。
- Starter では「review-pdfmaker」コマンドや「review-epubmaker」コマンドが使えません（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。

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。

【この章の内容】

2.1 用語の説明 10
2.2 フォルダとファイルの説明 11
2.3 原稿の追加と変更 12
2.4 基本的な記法 17
2.5 印刷用 PDF と電子用 PDF 27
2.6 高速プレビュー 29

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れます。

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もできます。

[*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。

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に必要となりますが、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/

LaTeX で使うスタイルファイルが置かれるフォルダです。

sty/mystyle.sty

ユーザが独自に LaTeX 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。LaTeX 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/starter.sty

Starter 独自のスタイルファイルです。ここに書かれた LaTeX 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-headline.sty

章 (Chapter) や節 (Section) や項 (Subsection) の LaTeX マクロが定義されたファイルです。

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、何でもいいので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、Mac なら Terminal.app[*2]、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

見ろ、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

原稿ファイルを「contents/」に置くよう設定している場合、もし原稿ファイル (*.re) 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[*3]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）[*4]。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。

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。

[*2] 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。
[*3] 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。
[*4] 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いけないため。

♣ 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
```

原稿ファイルには番号をつけるべき？

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[*5]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。

[*5] 章 ID については「♣ 章 ID」(p.40) で説明します。

▼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
```

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[*6]。
- 大扉（タイトルページ）や奥付も自動的につけられます[*7]。

[*6] 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。
[*7] 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。

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。

♣ コンパイルエラーになったら

PDF ファイルを生成するときにエラーになったら、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「`@<img>{}`」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

詳しくは「5.1 コンパイルエラー」(p.110) を見てください。

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「`@<strong>{...}`」または「`@<B>{...}`」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.34) 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.35) 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.38) 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」(p.47) 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[*8]、次の行からインデントします[*9]。

[*8] 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。
[*9] 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。

▼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「3.5 用語リスト」(p.48) を参照してください。

♣ 太字と強調

太字は「@<b>{...}」で囲み、強調は「@<B>{...}」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「@<B>{...}」または「@{...}」を使い、「@<b>{...}」は使わないでください。

強調や太字などテキストの装飾についての詳細は「3.7 強調と装飾」(p.51) を参照してください。

インライン命令

「@<B>{...}」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.49) を見てください。

♣ 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

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

第 1 引数も第 2 引数も、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[][]{」または「//list{」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。たとえば次の例では、追加した箇所を「@<b>{...}」で[*10]、削除した箇所を「@<del>{...}」で表しています。

[*10] プログラムリストの中では、「@<B>{}」（強調）ではなく「@<b>{}」（太字）を使ってください。なぜなら、「@<B>{}」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。

▼サンプル

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

プログラムコードについての詳細は「3.8 プログラムリスト」(p.53) 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.50) を見てください。

♣ 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「3.10 ターミナル」(p.62) 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。

▼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[*11]。

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

[*11] このように脚注文を書きます。

▼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「3.14 画像」(p.71) 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。

ノート本文には箇条書きやプログラムコードを埋め込めます。詳しくは「3.12 ノート」(p.64) を参照してください。

♣ 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。詳しくは「3.15 表」(p.77) 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「3.16 数式」(p.81) 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、環境変数「$STARTER_TARGET」を設定することで印刷用と電子用とを切り替えます。

▼macOS や Linux の場合[*12]

```
$ 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。

または、「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）に設定してもいいです。

[*12] 「pbook」は「printing book」、「ebook」は「electronic book」を表します。

ただし、この設定よりも環境変数「$STARTER_TARGET」のほうが優先されます。設定ファイルを変更しても切り替わらなくて困った場合は、環境変数を未設定に戻しましょう。

▼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）。
電子用では、カラーが使われます[*13]。

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[*14]。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。

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[*15]。

またこれらの違いに加えて、印刷用 PDF ファイルにはノンブルをつける必要があります。詳しくは次で説明します。

♣ ノンブル

「ノンブル」とは全ページにつける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。

- ページ番号は目次と本文で番号が連続してなかったり、空白ページにはついて

[*13] 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。
[*14] 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。
[*15] 表紙のつけ方は「♣ [PDF] 表紙を指定する」(p.104) を見てください。

なかったりするので、通し番号にはなっていません。ノンブルは全ページを通じて連続した番号になっています。

- ページ番号は読者のためにあるので、ページ内で目につくところに置かれます。ノンブルは印刷所だけが分かればいいので（ページの順番を確認するため）、ページ内で目立たない場所に小さく置かれます。

詳しくは「ノンブル qiita.com」で Google 検索してください。

印刷所に入稿する PDF ファイルには、ノンブルが必要になることがあります。ノンブルが必要かどうかは印刷所によって異なり、たとえば「日光企画」なら必須、「ねこのしっぽ」なら必須ではありません。詳しくは入稿先の印刷所に聞いてください。

Starter では、PDF ファイルにノンブルをつける機能が用意されています。

▼PDF ファイルにノンブルをつける

```
$ rake pdf              ← PDF ファイルを生成する
$ rake pdf:nombre       ← PDF ファイルにノンブルをつける
```

Docker を使っている場合は、「`rake pdf:nombre`」のかわりに「`rake docker:pdf:nombre`」を使ってください。また「`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。

なお「PDFOperation[*16]」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、印刷所に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。

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。

[*16] https://kauplan.org/pdfoperation/

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、環境変数「$STARTER_CHAPTER」を設定するとその章 (Chapter) だけをコンパイルします。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。

▼例：03-syntax-faq.re だけをコンパイルする

```
$ 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

$ STARTER_CHAPTER=03-syntax rake pdf  ←これでもよい
```

このとき、他の章は無視されます。また表紙や目次や大扉や奥付も無視されます。

全体をコンパイルする場合は、「$STARTER_CHAPTER」を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
```

♣ 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。

この機能は、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、特に効果が高いです。

ドラフトモードにするには、config-starter.yml で「draft: true」を設定するか、または環境変数「$STARTER_DRAFT」に何らかの値を入れてください。

▼ドラフトモードにして PDF を生成する

```
$ export STARTER_DRAFT=1  ←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

$ unset STARTER_DRAFT     ←ドラフトモードを off にする
```

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」という場合は、「//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。

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 で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はないので、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.17) を見たほうがいいでしょう。

【この章の内容】

3.1 コメント 34
3.2 段落と改行と空行 35
3.3 見出し 38
3.4 箇条書き 47
3.5 用語リスト 48
3.6 インライン命令とブロック命令 49
3.7 強調と装飾 51
3.8 プログラムリスト 53
3.9 行番号 59
3.10 ターミナル 62
3.11 脚注 64
3.12 ノート 64
3.13 コラム 68
3.14 画像 71
3.15 表 77
3.16 数式 81
3.17 その他 82

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。

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。

♣ 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。

[*1] なおこれは日本語 LaTeX の仕様です。

▼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

♣ 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

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。

ただし段と小段はそ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

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。

♣ 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や、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.38)
「♣ 節や項を参照する」(p.41)

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.38)
「♣ 節や項を参照する」(p.41)

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>{}」を使うとたとえば『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[*2]。

▼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。

♣ 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
```

[*2]「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。

```
    - 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。終わりにも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テキスト

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

♣ 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）

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

▼表示結果

(A) 番号つき箇条書き（英数字が使える）
(B) 番号つき箇条書き（任意の文字列が使える）
(B-1) 番号つき箇条書き（入れ子にできる）
- 番号なし箇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。

3.5 用語リスト

HTML でいうところの「<dl><dt><dd>」は、「 : 」で始めて、次の行からインデントします[*3]。先頭の半角空白は入れるようにしましょう[*4]。

[*3] 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。
[*4] 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たほうがいいでしょう。

▼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。

なお説明文の中に箇条書きやプログラムコードを入れることはできません。

3.6　インライン命令とブロック命令

ここで少し話を変えて、命令の正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：「`@<B>{...}`」）
- 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、「`@<B>{...}`」のような形式の命令のことです。主に文章中に埋め込むために使います。

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。

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

3.7　強調と装飾

♣ 強調

文章の一部を強調するには「@<B>{...}」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「@<B>{...}」または「@{...}」を使ってください。

インライン命令

「@<B>{...}」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「@<b>{...}」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「@<b>{...}」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「@<B>{...}」ではなく「@<b>{...}」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、「@<B>{...}」だとゴシック体のフォントに変更してしまうからです。「@<b>{...}」はフォントを変更しないので、等幅フォントのままで太字になります。

♣ 装飾

テキストを装飾するには、表 3.1 のようなインライン命令を使います。

♣ 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 3.2）。

♣ 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 3.3）。

▼表 3.1: 装飾用のインライン命令

意味	入力	出力
太字	@<b>{テキスト}	**テキスト**
強調	@{テキスト}	**テキスト**
強調	@<B>{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	@<del>{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

▼表 3.2: 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

▼表 3.3: マーキング用のインライン命令

意味	入力	出力
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}	xxx

3.8 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、ラベル名のかわりに「?」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 sys›
›tem administrator.")
//}
```

▼表示結果

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

「@<b>{}」で太字にしたり、「@<del>{}」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 t›
›o system administrator.")}
@<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 sys›
›tem administrator.")
sys.stderr.write("Something error raised. Please contact to
system administrator.")
```

ただし折り返し箇所が日本語だと、折り返しはされるものの、折返し記号がつきません[*5]。あまりいい対策ではありませんが、折り返したい箇所に「@<foldhere>{}」を書くと折り返しされつつ折返し記号もつきます。

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 sys›
›tem administrator.")
//}
```

[*5] 原因と対策が分かる人いたらぜひ教えてください。

▼表示結果

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

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 sys
tem administrator.")
//}
```

▼表示結果

```
sys.stderr.write("Something error raised. Please contact to sys
tem 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);
}
```

♣ インデント

第 3 引数にたとえば「[indentwidth=4]」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。

▼サンプル

```
//list[][][indentwidth=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。

♣ 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次のようにします[*6]。

[*6] 参考：https://github.com/kmuto/review/issues/887

▼別ファイルのソースコード (source/fib3.rb) を読み込む方法

```
//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「`@<b>{...}`」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。

♣ 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。

`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、Re:VIEW Starter ではまだ対応していません。

3.9　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 }
```

♣ 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,linenowidth=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,linenowidth=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)
```

3.10　ターミナル

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。

「@<userinput>{}」を使うと、ユーザによる入力箇所を示せます[*7]。

▼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
```

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}
//}
```

[*7] 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。

▼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
```

3.11　脚注

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-123}。本文。

//footnote[fn-123][脚注文。脚注文。]
```

▼表示結果

本文。本文[*8]。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。

3.12　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

[*8] 脚注文。脚注文。

```
//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)
```

♣ 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.66) 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。

引数の詳しい仕様を表 3.4 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。

▼表 3.4: 「//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

♣ 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）
- 「//caution」（警告）
- 「//notice」（注意）

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（デザインは同一です）。

▼サンプル

```
//warning[破滅の呪文]{
破滅の呪文を唱えると、本建物は自動的に瓦解します。
ご使用には十分注意してください。
//}
```

▼表示結果

破滅の呪文

破滅の呪文を唱えると、本建物は自動的に瓦解します。ご使用には十分注意してください。

このデザインを変更するには、「♣ [PDF] 「//info」や「//warning」のデザインを変更する」(p.95)

なお「//note」と違い、参照用のラベルはつけられません。また LaTeX の制限により、これらのブロックの内部では脚注がうまく表示されないことがあります。

3.13 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[*9]。
- 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。

ただし明確な違いはないので、ノートをコラム代わりに使っても、あるいはその逆でも間違いではないです。

♣ 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):
```

[*9] ただし「[notoc]」オプションをつけた場合を除く。

```
    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

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[*10]。

♣ 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。

▼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

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[*11]。

[*10] 脚注の文章。

[*11] これは LaTeX 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

現在のところ、これは仕様です。あきらめてください。

♣ ノートとコラムの使い分け

ノートとコラムは、次のように使い分けるといいでしょう。

- 基本的には、ノートを使う。
- 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。

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場合があるので、横着をせずに明示的にコラムを閉じましょう。

3.14 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][scale=1.0]`」を使います。

- 画像ファイルは「images/」フォルダに置きます[*12]。
- 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、「`@<img>{画像ファイル名}`」で参照できます。
- 第 3 引数には画像の表示幅を、本文幅に対する割合で指定します。たとえば「`[scale=1.0]`」なら本文幅いっぱい、「`[scale=0.5]`」なら本文幅の半分になります。

る必要があります。

[*12] 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。

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][scale=0.3]
```

表示例は図 3.1 を見てください。

▲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

画像ファイル名がラベルを兼ねるせいで、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「@<img>{}」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[*13]。

♣ 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」になり、画像ファイルを「images/02-tutorial/」に置けます[*14]（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[*15]を参照してください。

[*13] （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。

[*14] もちろん「images/」にも置けます。

[*15] 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

♣ 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 を見てください。

▲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。

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。

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[*16]。

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

[*16] 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。

画像はこれで表示するといいでしょう。

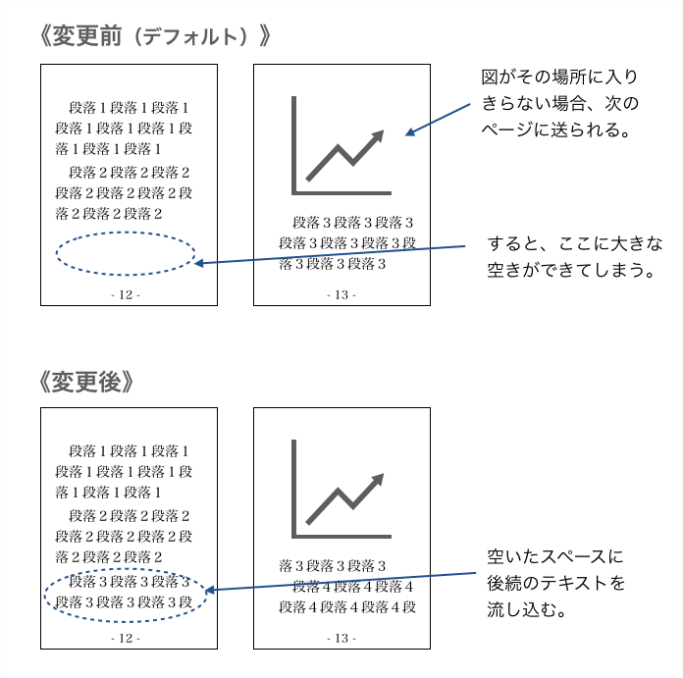
▼サンプル

```
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。



▲図 3.3: 「pos=H」（上）と「pos=h」（下）の違い

♣ 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きました。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

♣ 文章の途中に画像を読み込む

文章の途中に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 LaTeX 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。

3.15 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 3.5 を参照。

▼表 3.5: テーブルサンプル

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|...]」[*17]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

▼表 3.6: 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

♣ 罫線

「//tsize[|latex||l|r|l|c|]」の指定において、「|l|r|l|c|」の部分における「|」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[|latex|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
```

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

```
//}
```

▼表示結果

▼表 3.7: 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

♣ セル幅

セルの幅を指定するには、「//tsize[|latex|...]」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。テーブルが横にはみ出てしまう場合はセル幅を指定するといいでしょう。

▼サンプル

```
//tsize[|latex|l|p{70mm}|]
//table[tbl-sample4][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 text
BBB     text text text text text text text text text text text text
text text text text
//}
```

▼表示結果

♣ 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。

▼表 3.8: 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

▼サンプル

```
//tsize[|latex||l|ccc|]
//table[tbl-sample5][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

▼表 3.9: 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-sample6][セルの先頭が「@<code>{.}」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
//}
```

▼表示結果

▼表 3.10: 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

3.16 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[*18]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。
```

[*18] https://en.wikibooks.org/wiki/LaTeX/Mathematics

```
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、「@<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
- $\mathrm{y} = \mathrm{f(x)} + \mathrm{k}$ ← 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[*19]を参照してください。

3.17　その他

♣ URL、リンク

URL は「@<href>{URL}」と書きます。

[*19] https://github.com/kmuto/review/blob/master/doc/format.ja.md#tex-%E5%BC%8F

▼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[*20]

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>{}」の違いを比べて見ましょう。

▼サンプル

```
 * @<href>{https://twitter.com/_kauplan/, @_kauplan}
 * @<hlink>{https://twitter.com/_kauplan/, @_kauplan}
```

▼表示結果

- @_kauplan[*21]
- @_kauplan

♣ 引用

引用は「//quote{ ... //}」で表します。

[*20] https://kauplan.org/reviewstarter/
[*21] https://twitter.com/_kauplan/

▼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>{}
失われた大地との絆を結び、ついに人々を清浄の地に導かん。
//}
```

▼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。
> 失われた大地との絆を結び、ついに人々を清浄の地に導かん。

♣ 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

約束（エクス）された勝利（カリバー）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>{}」[*22]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： @<b>{テキスト}

強調されない： @@<nop>{}<b>{テキスト}
```

▼表示結果

強調される： **テキスト**
強調されない： @<b>{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
```

[*22] 「nop」は「No Operation」の略です。

```
$ 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。

♣ 改ページ

「//clearpage」で強制的に改ページできます。

♣ 特殊文字

- 「@<hearts>{}」で「♡」が表示できます。
- 「@<TeX>{}」で「TEX」が表示できます。
- 「@<LaTeX>{}」で「LATEX」が表示できます。

♣ 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
```

♣ 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

♣ 生データ

LaTeX 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 や ePub のときだけ「`//embed[html]{...//}`」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]{
<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

♣ 索引

（未執筆。Re:VIEW の Wiki[*23]を参照してください。）

♣ 参考文献

（未執筆。Re:VIEW の Wiki[*24]を参照してください。）

♣ 単語展開

（未執筆。Re:VIEW の Wiki[*25]を参照してください。）

[*23] https://github.com/kmuto/review/blob/master/doc/makeindex.ja.md

[*24] 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

[*25] https://github.com/kmuto/review/blob/master/doc/format.ja.md#%E5%8D%98%E8%AA%9E%E3%83%95%E3%82%A1%E3%82%A4%E3%83%AB%E3%81%AE%E5%B1%95%E9%96%8B

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。

【この章の内容】

4.1 命令 90
4.2 ページと本文 93
4.3 フォント 96
4.4 プログラムコード 97
4.5 見出し 99
4.6 本 104
4.7 Rake タスク 106

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.definline :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
    #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。

▼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(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

  ## HTML（ePub）用の定義
  class HTMLBuilder
    ## 入れ子を許す場合
    def on_blockquote(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[*1]。

▼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[*2]。

[*1] 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。

[*2] 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。

▼ 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.140) を見てください。

♣ [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
\addtolength{\textheight}{2Cvs}    % 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] 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@{03-syntax|subsec-miniblock}を見てください）。

これらのデザインを変更するには、「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」

```
%% 「//warning」のデザインを変更する
\renewenvironment{starterwarning}[1]{%
  \addvspace{1.0\baselineskip}% 1行分空ける
  \begin{oframed}%              % 枠線で囲む
    \ifempty{#1}\else%          % 引数（タイトル）があれば
      \noindent%                % 字下げせずに
      {\headfont\large%         % 太字のゴシック体で大きめに表示
         《警告》#1}%           % 先頭に 《警告》 をつける
      \par\smallskip%           % 縦方向に少しスペースを空ける
    \fi%
}{%
  \end{oframed}%                % 枠線による囲みの終わり
  \addvspace{1.0\baselineskip}% 1行分空ける
}
```

次はもっと派手なデザインのサンプルです。どんなデザインかは自分で確かめてください。

▼ファイル「sty/mystyle.sty」

```
%% 「//info」のデザインを変更する
\usepackage{ascmac}
\renewenvironment{starterinfo}[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[*3]。

▼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。

♣ [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。

[*3] 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。

▼表 4.2: 商業誌のフォントサイズと本文幅

書籍名（出版社）	各種サイズ
『リーダブルコード』（オライリー・ジャパン）	A5、9pt、38 文字
『ゼロから作る Deep Learning』（オライリー・ジャパン）	A5、9pt、38 文字
『達人に学ぶ SQL 徹底指南書 第 2 版』（翔泳社）	A5、9pt、38 文字
『アジャイル・サムライ』（オーム社）	A5、10pt、36 文字
『オブジェクト指向 UI デザイン』（技術評論社）	A5、9pt、35 文字
『現場で役立つシステム設計の原則』（技術評論社）	A5、10pt、33 文字
『独習プログラマー』（日経 BP 社）	A5、10pt、34 文字

▼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
  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。

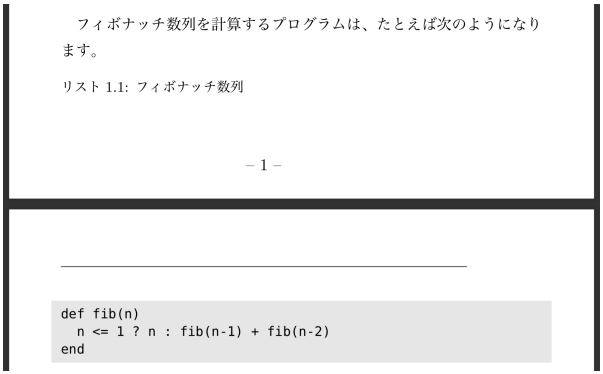
▼ファイル「sty/mystyle.sty」

```
%% 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「4.6zh」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 4.6 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。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・つけないを設定するには、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いだときは、枠線があった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.136)を見てください。

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「capter_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_oneline: 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
```

指定できる値は「config-starter.yml」の中に書かれています。

♣ 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。

♣ [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 4.2)。これを「章扉」といい、商業誌ではとてもよく見かけます。

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...
//}
```

▲図 4.2: 章扉（章の概要と目次つき）

```
//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] 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、「confit-starter.sty」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
```

♣ 項を目次に含める

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「項を目次に含める」です。もし

> 詳しくは「4.5.7 項を目次に含める」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「項を目次に含める」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「項を目次に含める」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.sty」で次のように定義されています。

▼ ファイル「sty/starter.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] 表紙を指定する

大扉（タイトルページ）ではなく、本の表紙を指定するには、「config.yml」の最後のほうにある設定項目「coverpdf_files:」に PDF ファイル名を指定します。

▼ファイル「config.yml」

```
  coverpdf_files: [cover_a5.pdf]          ←表紙
  backcoverpdf_files: []                  ←裏表紙
  coverpdf_option: "offset=-0.0mm 0.0mm" ←位置を微調整
```

ポイントは次の通りです。

- 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。
- PDF ファイルは、「images/」フォルダ直下に置いてください。
- 複数の PDF ファイルを指定するには、「[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。
- 裏表紙があれば「backcoverpdf_files:」に指定ます。

- 位置がずれる[*4]場合は、「coverpdf_option:」の値を調整してください。

なお表紙がつくのは、電子用 PDF ファイルの場合のみです。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.27) を参照してください。

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] 大扉のデザインを変更する

大扉（タイトルページ）のデザインは、「sty/mytitlepage.sty」で定義されています。大扉のデザインを変更すには、このファイルを変更してください。変更する前に、バックアップを取っておくといいでしょう[*5]。

```
$ cp sty/mytitlepage.sty sty/mytitlepage.sty.original
$ vi sty/mytitlepage.sty
```

♣ [PDF] 注意書きの文章を変更する

大扉（タイトルページ）の裏のページには、免責事項や商標に関する注意書きが書かれています。この文言を変更したい場合は、「sty/titlepage.sty」を編集してください。

♣ [PDF] 奥付のデザインを変更する

奥付のデザインは、「sty/mycolophon.sty」で定義されています。奥付のデザインを変更するには、このファイルを変更してください。変更する前に、バックアップを

[*4] LaTeX に詳しい人向け：取り込んだ PDF ファイルの位置がずれるのは、フォントサイズが 10pt ではないときです。このとき jsbook.cls では\mag の値を 1.0 以外に変更するので、それが pdfpages パッケージに影響します。一時的に\mag を 1.0 にして PDF ファイルを取り込む方法は分かりませんでした。

[*5] もちろん、Git 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。

取っておくといいでしょう[*6]。

```
$ cp sty/mytitlepage.sty sty/mytitlepage.sty.original
$ vi sty/mytitlepage.sty
```

4.7 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。

[*6] もちろん、Git 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。

▼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 (Frequentry Asked Question, FAQ) を紹介します。

【この章の内容】

5.1 コンパイルエラー 110
5.2 本文 111
5.3 プログラムコードとターミナル 112
5.4 その他 115

5.1　コンパイルエラー

♣ コンパイルエラーが出たとき

PDF ファイル生成時にコンパイルエラーになったときの対処方法について説明します。

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「@<img>{}」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.140)を見てください。

♣ 文章中のプログラムコードに背景色をつけたい

「@<code>{...}」を使って文章中に埋め込んだ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LaTeX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

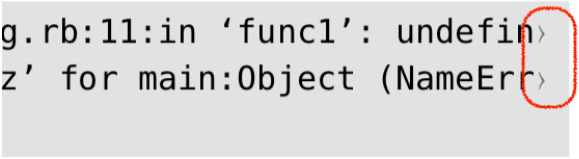
♣ プログラムコードの見た目が崩れる

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と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らず、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_wide: 0.0mm`」を、たとえば「`0.3mm`」に設定してください（値は各自

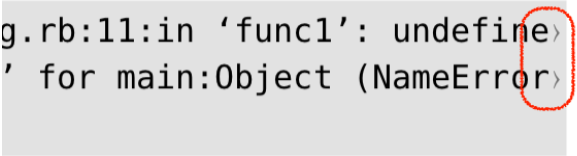
で調整してください）。

▼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_waiden:   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）。



▲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

♣ 半角文字の幅を全角文字の半分にしたい

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、以下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。

5.4 その他

♣ LaTeX 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 LaTeX 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。想像がつくと思いますが、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。

♣ 高度なカスタマイズ

設定ファイルや LaTeX 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。

[*1] Re:VIEW ではコードハイライトができるそうです。

- 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。

【この章の内容】

6.1 まえがき 120
6.2 初心者向け入門書 123
6.3 箇条書き 125
6.4 本文 128
6.5 見出し 130
6.6 プログラムコード 131
6.7 図表 138
6.8 ページデザイン 140
6.9 大扉 142
6.10 奥付 143

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。

▲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。

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」に本の対象読者が書かれてあると、本を手にとった人は自分がその本の対象読者かどうか、判断しやすいです。

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う。

- 「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。

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。

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と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.96) 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.93) を参照してください。

[*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。

♣ 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。

▲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。

▲図 6.3: Starter では節 (Section) と項 (Subsection) が見分けやすい

♣ 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.101) 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ますが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は成り立つので、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

番号つきの箇条書きは、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、番号つ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

ラベルつきの箇条書きは、一見すると番号つ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、ラベルつきの箇条書きは「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、項目の順序は重要では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つ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。

番号つき箇条書きとラベルつき箇条書きは、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、後者は意味がないか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トトロからであり、ナウシカとラピュタは厳密にはスタジオジブリの製作ではないのです。
```

▼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
- となりのトトロ

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トトロからであり、ナウシカとラピュタは厳密にはスタジオジブリの製作ではないのです。

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、段落の途中に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
 * となりのトトロ

しかしスタジオジブリが発足したのは実はトトロからであり、ナウシカとラピュタは厳密にはスタジオジブリの製作ではないのです。
```

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「@<b>{...}」ではなく「@<B>{...}」または「@{...}」を使ってください。

▼サンプル

```
本文本文@<B>{強調}本文本文。@<balloon>{ゴシック体なので望ましい}

本文本文@<b>{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よう、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、小説やエッセーや漫画では「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。

> その追尾弾（ハウンド）は相手を動かすための追尾弾（ハウンド）なのよ (p.132)

[*3] 『ワールドトリガー』22 巻（葦原大介、集英社、2020 年）

> だとすると、雨取(あまとり)ちゃんは人を狙って撃てないのか？ (p.133)

> 今の千佳(ちか)は間違いなくちゃんと戦う意志を持ってますよ (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

(TODO)

♣ 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

(TODO)

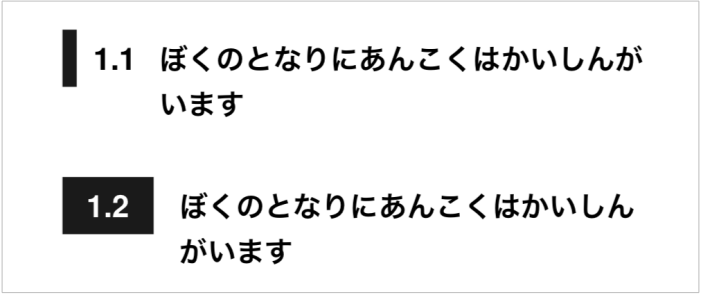
6.5 見出し

♣ 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。

タイトルが長い節 (Section) がある場合は、節タイトルに下線も背景色も使わないものがいいでしょう。具体的には次のどちらかを選びましょう。

- 「`leftline`」… 節タイトル左に縦線を引く（図 6.4 上）
- 「`numbox`」… 節番号を白ヌキ文字にする（図 6.4 下）



▲図 6.4: タイトルが複数行でも崩れない節デザインの例

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.100) を参照してください。

♣ 項を参照するなら項番号をつける

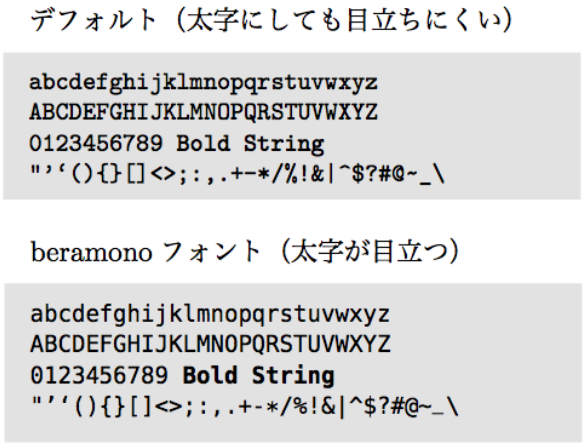
「@<secref>{}」や「@<hd>{}」で項 (Subsection) を参照するなら、項にも番号をつけましょう。番号のついていない見出しを参照するのは止めておいたほうがいいでしょう。

項に番号をつける方法は、「♣ 項に番号をつける」(p.102) を参照してください。

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。



▲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.98) 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[*4]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。

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 でも 10pt でも、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 でも 11pt でも、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_fontsize:`」と「`terminal_fontsize:`」で行われています。本文が A5 サイズ 10pt または B5 サイズ 11pt ならこれらが「`small`」に設定され、A5 サイズ 9pt または B5 サイズ 10pt なら「`normal`」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 10 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);
    }
```

[*4] 念のために説明すると、「8pt 以下」には 8pt も含まれます。

```
}
```

なおプログラムコードを太字にするときは、「@<B>{}」ではなく「@<b>{}」を使ってください。「@<B>{}」だと等幅フォントのはずがゴシック体に変わってしまいます。「@<b>{}」だと太字にするだけなので等幅フォントのままで変わりません。

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[*5]。

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。

♣ 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、削除した行に取り消し線を引き、追加した行を太字で表示すると、差分が分かりやすくなります。

たとえば次のようなサンプルコードがあるとします。

[*5] 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。

▼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

長い行を折り返したとき、折り返した箇所が分かるような表示が望ましいです。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っ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で、行番号がプログラムコードよりも目立たちません。

```
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")
```

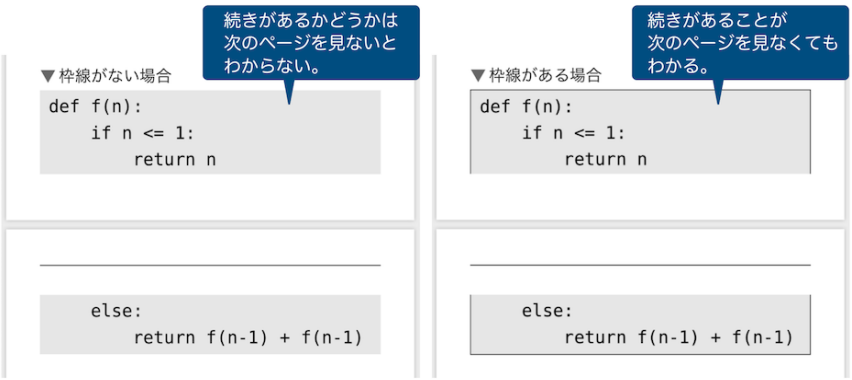
この例を見ると、折り返した行が左端には行かず、行番号の表示スペースを避けて折り返されています。これが、行番号つき折り返しの望ましい姿です。

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が、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[][][indentwidth=4]{ ... //}`」のようにすることで、インデントごとに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「`@<img>{ファイル名}`」、表なら「`@<table>{ラベル}`」で参照します。

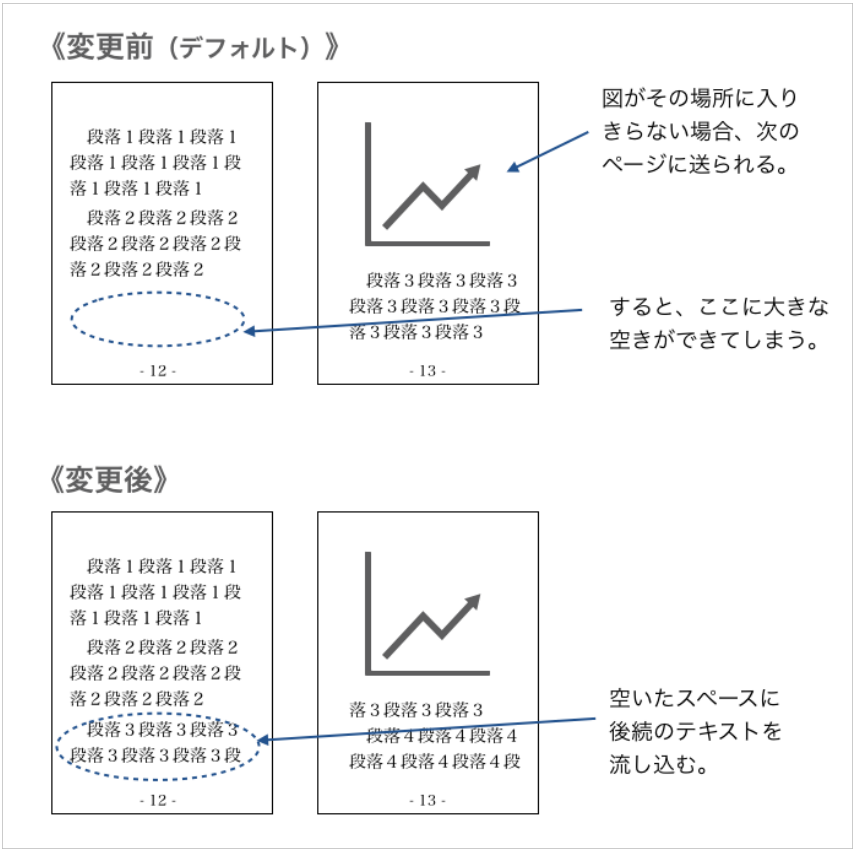
♣ 表のカラム幅を指定する

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-jqqu9][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。失われた
```



▲図 6.7: 後続の本文を現在位置に流し込むかどうか

```
大地との絆を結び、ついに人々を清浄の地に導かん。
ゴンドアの谷　リーテ・ラトバリタ・ウルス・アリアロス・バル・ネロリール
//}
```

▼表示結果

♣ 表のカラムが数値なら右寄せにする

表のカラム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

▼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。失われた大地との絆を結び、ついに人々を清浄の地に導かん。
ゴンドアの谷	リーテ・ラトバリタ・ウルス・アリアロス・バル・ネロリール

きます。特にカラムが数値なら、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

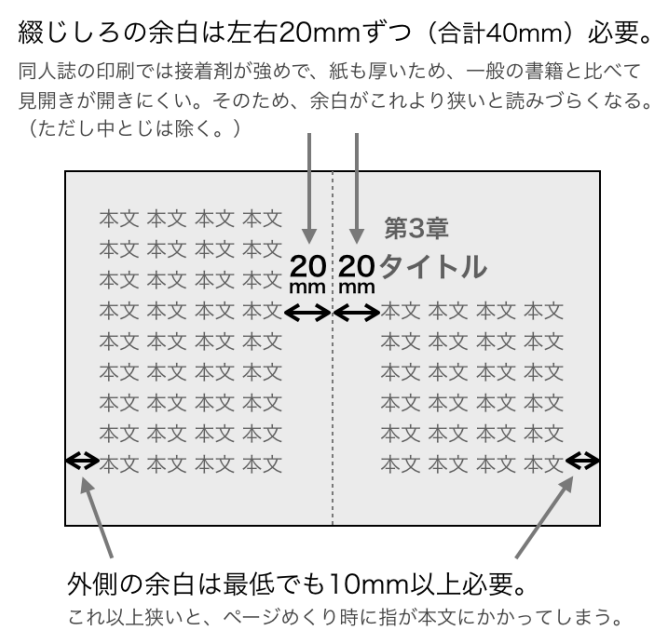
6.8 ページデザイン

♣ [PDF] 印刷用では左右の余白幅を充分にとる

紙の書籍では、左右いっぱ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 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

(TODO)

♣ 電子書籍ではページ番号の位置を揃える

(TODO)

♣ 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しています。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ことを検討しましょう。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.96)を参照してください。

6.9 大扉

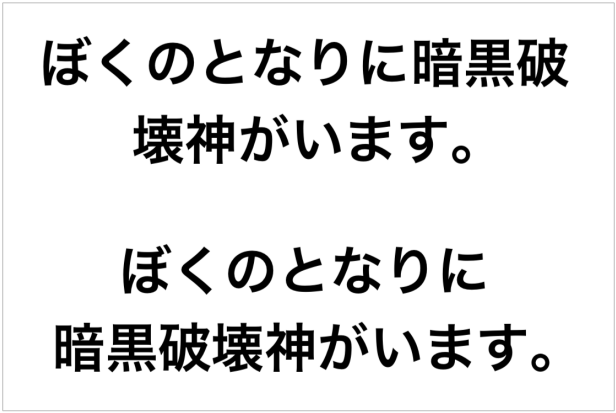
♣ 長いタイトルでは改行箇所を明示する

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）。

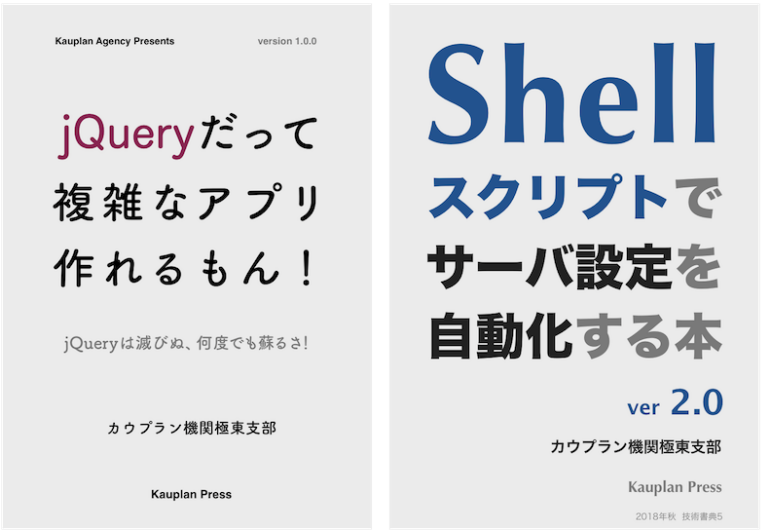
これを防ぐために、Starter ではタイトルを複数行で指定できるようになっています。この場合、大扉（タイトルページ）ではタイトルが 1 行ずつ表示されるので、自然な位置を指定して改行できます（図 6.9 下）。

♣ 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）。



▲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）



▲図 6.10: Keynote で作成した大扉の例

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[*6]。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るようになっています。

♣ 奥付に更新履歴とイベント名を記載する

奥付には、本の更新履歴が記載されるのが一般的です。

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[*7]。

♣ 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。

[*6] 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。

[*7] 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。

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。

（TODO）
まだ書かれてないので、かわりに Qiia の記事を参照してください。

- 『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）』(Qiita)
  https://qiita.com/kauplan/items/dd8dd0f4a6e0eb539a98
- 『Re:VIEW Starter の新機能（2019 年冬）』(Qitta)
  https://qiita.com/kauplan/items/e36edd7900498e231aaf

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れます。

[*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。

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もできます。

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に必要となりますが、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

LaTeX 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 LaTeX 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。LaTeX 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 LaTeX 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.eruby

プログラムコードやターミナルのカスタマイズ用です。

sty/starter-color.sty.eruby

[*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。

色のカスタマイズ用です。

sty/starter-font.sty.eruby

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 LaTeX マクロが定義されたファイルです。

sty/starter-note.sty.eruby

ノートブロックのカスタマイズ用です。

sty/starter-section.sty.eruby

以前の、章や節の LaTeX 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-toc.sty.eruby

目次のカスタマイズ用です。

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。

C.1 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]。具体的は修正方法は Qiita の記事[*3]に書いています。読

[*1] https://www.slideshare.net/KazutoshiKashimoto/nagoya0927-release の p.21 と p.22。

[*2] 簡単に書いてますけど、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。今でも恨んでます。この苦労を知らずに「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バットの刑を与えてやりたいくらいです。

[*3] https://qiita.com/kauplan/items/01dee0249802711d30a6

▲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

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。

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。

C.2 開発の転機

Starter は、当初は Re:VIEW との違いが大きくならないよう、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

[*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。

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。

C.3 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てから 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。

[*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ないので、デメリットはありません。

「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行う。
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に置き換えることになるでしょう。

C.4　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??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のかもしれませんが、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[*7]。

♣ バグ対応

今まで、Re:VIEW には約 20 個ぐらいのバグ報告や 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
  CC CC
```

▼ 表示結果

- AA AABB BBCC CC

*6 https://www.w3.org/TR/jlreq/ja/

*7 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。

「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

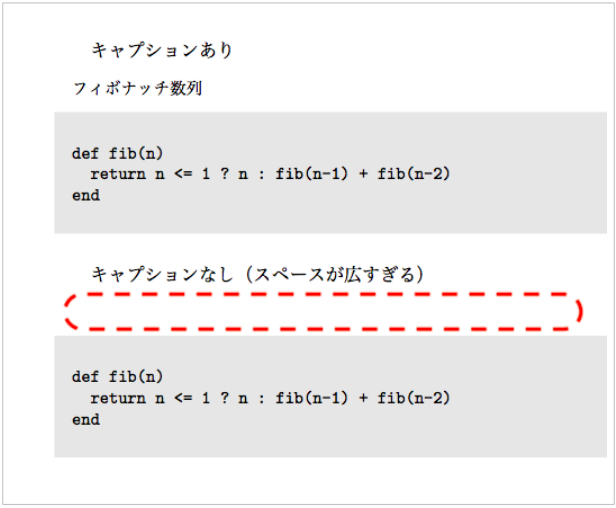
昔の Re:VIEW では、「//list」においてキャプション（説明文）を指定しなくても言語指定をすると、本文とプログラムリストの間が大きく空いてしまうというバグがありました。たとえばこのように書くと：

▼ 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ない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と逆に感心しました。

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。

C.5　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 Bibliostyle をサポートしたいと考えています。ただし Bibliostyle 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、Bibliostyle 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、カラー化がいっそう進むでしょう。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

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）

Re:VIEW Starter ユーザーズガイド

インストールからカスタマイズまで

2020 年 7 月 7 日　ver 1.0

著　者　カウプラン機関極東支部
印刷所　○○印刷所

© 2020 カウプラン機関極東支部

(powered by Re:VIEW Starter)