Enumitem パッケージ

@monaqa

目次

1. パッケージの概要

enumitem は SATYSFI にて豊富な箇条書きや番号付きのリストを提供するパッケージです。標準にもすでに itemize という名前のパッケージが用意されているものの、本パッケージではより自由度の高い箇条書きを提供します。具体的には、以下のような箇条書きを書くことができます：

- デフォルトで豊富なスタイルのラベルを選択できる
- 番号付き箇条書き環境をネストさせることができる
- ネストごとに箇条書きのスタイルを変更できる
- 箇条書きのラベルの体裁を項目ごとに変更できる
- 定義リストを作成できる
- ユーザ自身がスタイルを拡張できる

本ドキュメントは enumitem パッケージ v3.0.0 の仕様及び使い方を述べたものです。旧バージョン、すなわちメジャーバージョンが 0, 1, 2 のいずれかであるものとの互換性は無く、たとえコマンド名が同じであっても引数のルールが異なることなどがあります。バージョンの違いに注意してください。

1.1. 用語集

ここでは以下のような用語を用います。

箇条書き　いくつかの単語や文章、段落などを分けて書き並べたリストのこと。ここでは本文と独立の段落で組むものを扱う。

項目　箇条書きを構成する要素。

ラベル　箇条書きの開始を表すために先頭につける記号や文字の列。番号付きの箇条書きであれば “1.” や “(1)” といった文字列を指す。

番号付き箇条書き　番号や順序のわかる記号を用いてラベルがふられた箇条書き。

番号無し箇条書き　順序のない記号など用いてラベルがふられた箇条書き。

2. enumitem をはじめよう

2.1. enumitem パッケージのインストール

enumitem パッケージは Satyrographos によってインストールすることができます。

```
opam install satysfi-enumitem
satyrographos install
```

opam list satysfi-enumitem などのコマンドでバージョン 3 系がインストールされていることを確認したら、次に進みましょう。

2.2. パッケージの読み込み

SATYSFI の文書ファイルやヘッダファイルで外部のパッケージを用いるには以下のようにします。

```
@require: enumitem/enumitem
```

さらに、EnumitemFormatAlias というモジュールも open しておきましょう。 これは enumitem パッケージで定義されているモジュールの一つであり、箇条書きの体裁を変更するための関数、定義リスト専用のコマンドなど便利な機能がまとまっています。まとめると、文書ファイルの場合は以下のような文言をファイルに書けばよいということになります。「本文」のところは、各クラスファイルの書き方に従ってください。

```
@require: enumitem/enumitem

open EnumitemFormatAlias

in

（本文）
```

準備ができたら、本文のどこかに以下のように書いてみましょう。

```
+listing{
  * foo
  * bar
    ** barfoo
    ** barbar
```

```
  * baz
}
```

もし正常にパッケージが読み込まれていれば、以下のように出力されるはずです。

- foo
- bar
    - barfoo
    - barbar
- baz

2.3. 基本的なコマンドの使用

enumitem は +listing 及び +enumerate コマンドを提供します。 標準の itemize パッケージを使ったことがある人にとっては馴染み深いでしょう。 使い方は標準のコマンドと（オプション引数を除き）ほぼ変わりません。

```
+listing{
  * hoge
  * fuga
    ** fuga1
      *** fuga11
      *** fuga12
    ** fuga2
}
```

- hoge
- fuga
    - fuga1
        - fuga11
        - fuga12
    - fuga2

```
+enumerate{
  * hoge
```

```
  * fuga
    ** fuga1
      *** fuga11
      *** fuga12
    ** fuga2
}
```

1. hoge
2. fuga
   (a) fuga1
      (i) fuga11
      (ii) fuga12
   (b) fuga2

3. enumitem の設計思想

enumitem v3.0.0 は以下の思想を元に設計されています。

- $\mathrm{SAT_YSF_I}$ の文法に備わっている箇条書き用の構文は極力尊重する。
- 箇条書き用の構文でカバーできない機能については、専用のコマンドで対応する。
- 関数を用いてラベルの体裁を指定することで、ユーザによる拡張ができるようにする。

3.1. 2 種類の表現方法

簡潔に書ける箇条書きと自由度の高い箇条書きを両立させるため、enumitem パッケージでは箇条書きの木構造を表現する 2 種類の表現方法が用意されています。

- aaa
- bbb
  - bbb の子 1
  - bbb の子 2
    - bbb の子 2 の子 1
- ccc

- ccc の子 1
- ccc の子 2

1 つは SATYSF$_{\text{I}}$ の箇条書きを表す専用の構文をそのまま用いてネストを表現する方法。こちらは標準の itemize パッケージでも採用されているインターフェースであり、簡単に記述できるのが利点です。ただし、設定できる項目の自由度はあまり高くありません。便宜上、ここではこちらを**糖衣構文ベースの記法**と呼びます。

```
+listing?:(listing-default-label){
  * aaa
  * bbb
    ** bbb の子 1
    ** bbb の子 2
      *** bbb の子 2 の子 1
  * ccc
    ** ccc の子 1
    ** ccc の子 2
}
```

なお、+listing のオプション引数のデフォルトは listing-default-label であるため、上のコードの ?:(listing-default-label) に相当する箇所は省略することができます（次の例との比較のためあえてつけています）。

もう 1 つは SATYSF$_{\text{I}}$ の箇条書き専用構文を用いることなく、+itemize 及び +item のみを用いてブロックテキストベースで箇条書きのネストを表現する方法。星印 * だけでアイテムの始まりを示すことができた糖衣構文ベースの記法とは異なり、項目のたびに +item を書く必要があります。記法としては冗長になるものの、箇条書きを記述する際の自由度が高いというメリットがあります。こちらを**ブロックベースの記法**と呼びます。

```
+itemize(listing-default-label)<
  +item{aaa}<>
  +item{bbb}<
    +item{bbb の子 1}<>
    +item{bbb の子 2}<
      +item{bbb の子 2 の子 1}<>
    >
  >
  +item{ccc}<
```

```
    +item{ccc の子 1}<>
    +item{ccc の子 2}<>
  >
>
```

ここで示した 2 つのコードは、どちらも最終的には同様の結果となることに注意してください。実は 1 番目の糖衣構文ベースの記法は、まず内部でブロックベースの記法へと展開されてから処理されます。

3.2. ブロックベース記法の使いみち

上で示したコードは、糖衣構文ベースの記法でもブロックベースの記法でも、どちらでも実現可能な箇条書きでした。その場合は、より簡潔に書ける糖衣構文ベースの記法を用いるべきでしょう。 ブロックベースの記法の利点は自由度の高さにあります。 つまり、 糖衣構文ベースでは実現できない箇条書きも表現することができるのです。

1 つは箇条書きの途中で項目の体裁を変更すること。

```
+itemize(listing-default-label)<
  +item{aaa}<>
  +item?:(raw-arabic){bbb}<
    +item{bbb の子 1}<>
    +item{bbb の子 2}<
      +item{bbb の子 2 の子 1}<>
    >
  >
  +item?:(dot-arabic-rec){ccc}<
    +item{ccc の子 1}<>
    +item{ccc の子 2}<>
  >
>
```

- aaa

 2 bbb

  - bbb の子 1
  - bbb の子 2
    - bbb の子 2 の子 1

3. ccc

- ccc の子 1
- ccc の子 2

```
+itemize(listing-default-label)<
  +item{aaa}<>
  +item({bbb})?:(raw-arabic)<
    +item{bbb の子 1}<>
    +item{bbb の子 2}<
      +item{bbb の子 2 の子 1}<>
    >
  >
  +item({ccc})?:(dot-arabic-rec)<
    +item{ccc の子 1}<>
    +item{ccc の子 2}<>
  >
>
```

- aaa
- bbb
  1. bbb の子 1
  2. bbb の子 2
     1. bbb の子 2 の子 1
- ccc
  3.1. ccc の子 1
  3.2. ccc の子 2

もう 1 つは、項目の下に任意のブロックテキストを配置できること。

```
+itemize(listing-default-label)<
  +item{ここは最初の段落です。ラベルは最初の段落の冒頭に付きます。}<
    +pn{
      ここは 2 番目の段落です。
      2 番目以降であれば自由にブロックテキストを挿入できます。
      以下はコードブロックを挿入する例。
    }
    +code(`aaa`);
```

```
    +item{このように、途中で箇条書きを挟むことができます。}<
      +item{当然ネストさせることもできます。}<>
    >
    +pn{
      箇条書きを挟んだ後、元の階層に戻って再び段落を再開することができます。
    }
  >
>
```

- ここは最初の段落です。ラベルは最初の段落の冒頭に付きます。

  ここは 2 番目の段落です。2 番目以降であれば自由にブロックテキストを挿入できます。以下はコードブロックを挿入する例。

  ```
  aaa
  ```

  - このように、途中で箇条書きを挟むことができます。
    - 当然ネストさせることもできます。

  箇条書きを挟んだ後、元の階層に戻って再び段落を再開することができます。

3.3. 2 つの記法の併用

2 つの記法は組み合わせて使用することもできます。

```
+itemize(listing-default-label)<
  +item{aaa}<>
  +item{bbb}<
    +sublist(listing-default-label){
      * bbb の子 1
      * bbb の子 2
        ** bbb の子 2 の子 1
    }
  >
  +sublist(listing-default-label){
    * ccc
      ** ccc の子 1
      ** ccc の子 2
  }
>
```

```
+listing{
  * aaa
  * bbb
    \subitem(listing-default-label)<
      +item{bbb の子 1}<>
      +item{bbb の子 2}<
        +item{bbb の子 2 の子 1}<>
      >
    >
  * ccc
    \subitem(listing-default-label)<
      +item{ccc の子 1}<>
      +item{ccc の子 2}<>
    >
}
```

\sublist を用いれば、糖衣構文ベースの記法を（糖衣構文を直接使わず）ネストさせることができます。

```
+listing{
  * aaa
  * bbb
    \sublist(listing-default-label){
      * bbb の子 1
      * bbb の子 2
        \sublist(listing-default-label){
          * bbb の子 2 の子 1
        }
    }
  * ccc
    \sublist(listing-default-label){
      * ccc の子 1
      * ccc の子 2
    }
}
```

4. Gallery

4.1. ラベルの変更

+listing コマンドを用いて番号無し箇条書きを実現できることは既に見たとおりですが、オプション引数を用いるとラベルを変更することができます。以下の例はラベルに bullet 及び white-bullet を指定した箇条書きであり、ラベルがそれぞれ黒丸、白丸に置き換わっていることがわかります。

```
+listing?:(bullet){
  * 水馬、赤いな。ア、イ、ウ、エ、オ。
    ** 浮藻に小蝦もおよいでる。
}
+listing?:(white-bullet){
  * 水馬、赤いな。ア、イ、ウ、エ、オ。
    ** 浮藻に小蝦もおよいでる。
}
```

- 水馬、赤いな。ア、イ、ウ、エ、オ。
  - 浮藻に小蝦もおよいでる。
- 水馬、赤いな。ア、イ、ウ、エ、オ。
  - 浮藻に小蝦もおよいでる。

+enumerate コマンドも同様にオプション引数を用いてラベルを変更できます。

```
+enumerate?:(paren-alph){
  * 水馬、赤いな。ア、イ、ウ、エ、オ。
    ** 浮藻に小蝦もおよいでる。
  * 柿の木、栗の木。カ、キ、ク、ケ、コ。
    ** 啄木鳥こつこつ、枯れけやき。
  * 大角豆に醋をかけ、サ、シ、ス、セ、ソ。
    ** その魚淺瀬で刺しました。
}
```

(a) 水馬、赤いな。ア、イ、ウ、エ、オ。

(a) 浮藻に小蝦もおよいでる。

(b) 柿の木、栗の木。カ、キ、ク、ケ、コ。

(a) 啄木鳥こつこつ、枯れけやき。

(c) 大角豆に醋をかけ、サ、シ、ス、セ、ソ。

(a) その魚淺瀬で刺しました。

なお、+listing と +enumerate の違いはオプション引数のデフォルト値のみであるため、上の例で +listing?:(paren-alph){ ... } と書いても同じ結果となります。とはいえマークアップ上の視認性から、番号付き箇条書きには原則 +enumerate を使用するほうが望ましいでしょう。

もう少し自由度高くラベルを設定したい場合もあるでしょう。text-label 関数を用いると、任意のインラインテキストをラベルとして取り扱うことができます。

```
+listing?:(text-label {一. }){
  * 水馬、赤いな。ア、イ、ウ、エ、オ。
    ** 浮藻に小蝦もおよいでる。
  * 柿の木、栗の木。カ、キ、ク、ケ、コ。
    ** 啄木鳥こつこつ、枯れけやき。
  * 大角豆に醋をかけ、サ、シ、ス、セ、ソ。
    ** その魚淺瀬で刺しました。
}
```

一. 水馬、赤いな。ア、イ、ウ、エ、オ。

一. 浮藻に小蝦もおよいでる。

一. 柿の木、栗の木。カ、キ、ク、ケ、コ。

一. 啄木鳥こつこつ、枯れけやき。

一. 大角豆に醋をかけ、サ、シ、ス、セ、ソ。

一. その魚淺瀬で刺しました。

index-label 関数を用いると、自由度の高い番号付き箇条書きを生成することもできます。$\mathrm{SAT_{Y}SF_{I}}$ の記法に慣れていなければ少し複雑ですが、インデックスの値（整数値）を引数に取ってインラインテキストを返すような関数を与えることで対応するラベルを生成することができます。

```
+enumerate?:(
```

```
  let labelf idx =
    let it = (idx * idx) |> arabic |> embed-string in
    {\ #it;.\ }
  in
  index-label labelf
){
  * 水馬、赤いな。ア、イ、ウ、エ、オ。
    ** 浮藻に小蝦もおよいでる。
  * 柿の木、栗の木。カ、キ、ク、ケ、コ。
    ** 啄木鳥こつこつ、枯れけやき。
  * 大角豆に醋をかけ、サ、シ、ス、セ、ソ。
    ** その魚淺瀬で刺しました。
}
```

1. 水馬、赤いな。ア、イ、ウ、エ、オ。
    1. 浮藻に小蝦もおよいでる。
4. 柿の木、栗の木。カ、キ、ク、ケ、コ。
    1. 啄木鳥こつこつ、枯れけやき。
9. 大角豆に醋をかけ、サ、シ、ス、セ、ソ。
    1. その魚淺瀬で刺しました。

ここまで見てきた例はいずれもラベルが箇条書きのネストの深さに応じて変化しませんでした。しかし、親の項目には番号付きのラベルを、子の項目には番号無しのラベルを振るケースも珍しくありません。ネストの深さに応じてラベルを変化させたい場合は `with-depth` 関数を用います。`with-depth` 関数はラベルのリストを引数に取り、 項目のネストの深さが n である項目はリストの n 番目のラベルを用いて組まれます。

```
+enumerate?:(with-depth [paren-alph; text-label {続き：}]){
  * 水馬、赤いな。ア、イ、ウ、エ、オ。
    ** 浮藻に小蝦もおよいでる。
  * 柿の木、栗の木。カ、キ、ク、ケ、コ。
    ** 啄木鳥こつこつ、枯れけやき。
  * 大角豆に醋をかけ、サ、シ、ス、セ、ソ。
    ** その魚淺瀬で刺しました。
}
```

(a) 水馬、赤いな。ア、イ、ウ、エ、オ。

続き：浮藻に小蝦もおよいでる。

(b) 柿の木、栗の木。カ、キ、ク、ケ、コ。

続き：啄木鳥こつこつ、枯れけやき。

(c) 大角豆に醋をかけ、サ、シ、ス、セ、ソ。

続き：その魚淺瀬で刺しました。

```
+listing?:({| 一. | 二. | 三. |} |> List.map text-label |> with-depth){
  * 五十音
    ** 水馬、赤いな。ア、イ、ウ、エ、オ。
      *** 浮藻に小蝦もおよいでる。
    ** 柿の木、栗の木。カ、キ、ク、ケ、コ。
      *** 啄木鳥こつこつ、枯れけやき。
    ** 大角豆に醋をかけ、サ、シ、ス、セ、ソ。
      *** その魚淺瀬で刺しました。
}
```

一. 五十音

二. 水馬、赤いな。ア、イ、ウ、エ、オ。

三. 浮藻に小蝦もおよいでる。

二. 柿の木、栗の木。カ、キ、ク、ケ、コ。

三. 啄木鳥こつこつ、枯れけやき。

二. 大角豆に醋をかけ、サ、シ、ス、セ、ソ。

三. その魚淺瀬で刺しました。

4.2. 複雑な箇条書き

第3章で述べたとおり、複雑な箇条書きは `+listing` などの糖衣構文を用いるコマンドで表現しきれない場合があります。そのような場合は `+itemize` コマンドを用います。

```
+itemize(dot-arabic)<
  +item{ここはアラビア数字で番号付けされている。}<>
  +item{1 番目の必須引数に書いたものが項目となる。}<
    +item{ネストさせることも可能。}<>
```

```
    +item{2 番目の必須引数に書いたものが子項目となる。}<>
  >
  +item?:(paren-roman){ここはローマ数字。}<
    +item{1 つ目の必須引数の前を変えると、その項目のラベルのみ変わる。}<>
    +item{したがって、子の項目はローマ数字にならず、アラビア数字のまま。}<>
  >
  +item?:(paren-alph)({ここはアルファベット。})?:(bracket-arabic)<
    +item{1 つ目の必須引数の前を変えると、子の項目のラベルが変わる。}<>
    +item{今回の場合、ネストの中身は角括弧で囲まれたアラビア数字となる。}<
      +item{中身も角括弧で囲まれたアラビア数字。}<>
    >
  >
  +item{ここはアラビア数字。}<>
>
```

1. ここはアラビア数字で番号付けされている。
2. 1 番目の必須引数に書いたものが項目となる。
    1. ネストさせることも可能。
    2. 2 番目の必須引数に書いたものが子項目となる。

(iii) ここはローマ数字。

1. 1 つ目の必須引数の前を変えると、その項目のラベルのみ変わる。
2. したがって、子の項目はローマ数字にならず、アラビア数字のまま。

(d) ここはアルファベット。

[1] 1 つ目の必須引数の前を変えると、子の項目のラベルが変わる。

[2] 今回の場合、ネストの中身は角括弧で囲まれたアラビア数字となる。

[1] 中身も角括弧で囲まれたアラビア数字。

5. ここはアラビア数字。

項目の中に複数の段落を入れたいときにも `+itemize` と `+item` の組み合わせは有用です。

```
  +item{
    一つの項目に複数の段落を入れる例。
    `+item` の 1 番目の必須引数に入力されたものが段落となる。
```

```
  }<
    +p{
      しかし、2 番目の必須引数に `+p` コマンドなど通常のコマンドを入れると、
      段落をはじめとしたブロックを後続させることができる。
    }
    +p{
      段落は当然複数置くことができる。
    }
  >

  +item{
    別行立て数式を配置する例。
  }<
    +math(${y = a});

    +p{
      インラインテキスト内で別行立て数式を配置するコマンドも存在するが、
      マークアップの意味合いは若干異なり、こちらは改段落が行われる。
    }
  >
>
```

- 一つの項目に複数の段落を入れる例。+item の 1 番目の必須引数に入力されたものが段落となる。

  しかし、2 番目の必須引数に +p コマンドなど通常のコマンドを入れると、段落をはじめとしたブロックを後続させることができる。

  段落は当然複数置くことができる。

- 別行立て数式を配置する例。

$$y = a$$

  インラインテキスト内で別行立て数式を配置するコマンドも存在するが、マークアップの意味合いは若干異なり、こちらは改段落が行われる。

場合によっては、\sublist などを用いることで糖衣構文ベースで複雑な箇条書きを実現できるケースもあります。

```
+listing{
  * 番号付き箇条書きと番号なし箇条書きの混用。
```

```
    ** ここは番号無し箇条書き。
      \sublist(bullet){
      * ここは番号なし箇条書き。
      * ここは番号なし箇条書き。
      * ここは番号なし箇条書き。
    }
    ** ここは番号付き箇条書き。
      \sublist(dot-arabic){
      * ここは番号付き箇条書き。
      * ここは番号付き箇条書き。
      * ここは番号付き箇条書き。
    }
}
```

- 番号付き箇条書きと番号なし箇条書きの混用。
  - ここは番号無し箇条書き。
    - ここは番号なし箇条書き。
    - ここは番号なし箇条書き。
    - ここは番号なし箇条書き。
  - ここは番号付き箇条書き。
    1. ここは番号付き箇条書き。
    2. ここは番号付き箇条書き。
    3. ここは番号付き箇条書き。

4.3. 応用： To Do リスト

enumitem を用いて To Do リストを実現することもできます。こちらは少し前準備が必要となります。まずはプリアンブル（トップレベル）にて以下のコマンドを定義します。

```
let ooalign ib1 ib2 =
  let (wid, ht, dp) = get-natural-metrics ib1 in
  inline-graphics wid ht dp (fun pt -> [draw-text pt ib1; draw-text pt ib2])
let ib-box ctx =
  let ctx = ctx |> set-dominant-narrow-script Kana in
  read-inline ctx (string-unexplode [0x2713] |> embed-string)
let ib-checkmark ctx =
```

```
  let ctx = ctx |> set-dominant-narrow-script Kana in
  read-inline ctx (string-unexplode [0x2501] |> embed-string)

let-inline ctx \mark-todo = ib-box ctx
let-inline ctx \mark-done = ooalign (ib-box ctx) (ib-checkmark ctx)
```

`\mark-todo;` は未完了タスクを表すラベル「☐」を、`\mark-done;` は完了タスクを表すラベル「☑」を表しています。上では用意されているフォントのグリフを重ね打ちするなどして作成したものの、画像ファイルを読み込む、SATySFi のグラフィックス機能を用いて描画する、といった手法でラベルを用意することも可能です。

続いて、以下のように 3 種類のコマンドを定義します。

```
let-block +todo-list bt = '< +itemize(text-label {\mark-todo;\ })(bt); >
let-block +todo it = '< +item?:(text-label {\mark-todo;\ })(it)<> >
let-block ctx +done it =
  let ctx = ctx |> set-text-color (Color.gray 0.5) in
  read-block ctx '< +item?:(text-label {\mark-done;\ })(it)<> >
```

これらのコマンドを用いて、以下のように To Do リストを組むことができます。

```
+todo-list<
  +todo{ミルクを買う。}
  +todo{The \SATySFi;book を読む。}
  +done{\SATySFi; を完全に理解する。}
  +todo{課題を解く。}
>
```

☐ ミルクを買う。

☐ The SATySFibook を読む。

☑ SATySFi を完全に理解する。

☐ 課題を解く。

4.4. 応用：定義リスト

定義リストは箇条書きの中でも比較的複雑であり、ラベルに相当する部分にその項目が説明する用語名が入ります。つまり項目ごとにラベルの内容が変化する箇条書きと表現することもできます。幸い、今まで見たとおり `enumitem` パッケージにはそういった複雑な箇条書きを表現するだけの十分な機能が備わっています。

最も単純な定義リストは以下のようなものです。

```
+description<
  +ditem{用語 1}{ここに用語 1 の説明が入る。}< >
  +ditem{用語 2}{
    ここに用語 2 の説明が入る。用語 2 の説明は少し長い。
    用語 2 の説明は少し長い。用語 2 の説明は少し長い。用語 2 の説明は少し長い。
  }< >
  +ditem{ちょっと長めの用語 3}{
    ここに用語 3 の説明が入る。用語 3 の説明は少し長い。
    用語 3 の説明は少し長い。用語 3 の説明は少し長い。用語 3 の説明は少し長い。
  }<
    +pn{
    複数行にまたがる説明を入れることもできる。
    }
  >
>
```

用語 1　ここに用語 1 の説明が入る。

用語 2　ここに用語 2 の説明が入る。用語 2 の説明は少し長い。用語 2 の説明は少し長い。用語 2 の説明は少し長い。用語 2 の説明は少し長い。

ちょっと長めの用語 3　ここに用語 3 の説明が入る。用語 3 の説明は少し長い。用語 3 の説明は少し長い。用語 3 の説明は少し長い。用語 3 の説明は少し長い。

複数の段落にまたがる説明を入れることもできる。

+ditem は +item と似ているものの、追加で 1 番目にインラインテキストの必須引数を取り、そこに書かれたテキストをラベルの名前として表記することができます。

ラベルを太字にするなどして強調表示したいことはよくあります。+ditem のオプション引数で強調を表現することができます。

```
+description<
  +ditem?:(fun it -> {\emph{#it;}}){用語 1}{ここに用語 1 の説明が入る。}< >
  +ditem?:(fun it -> {\emph{#it;}}){ちょっと長めの用語 3}{
    ここに用語 3 の説明が入る。用語 3 の説明は少し長い。
    用語 3 の説明は少し長い。用語 3 の説明は少し長い。用語 3 の説明は少し長い。
  }< >
>
```

用語 *1*　ここに用語 1 の説明が入る。

ちょっと長めの用語 *3*　ここに用語 3 の説明が入る。用語 3 の説明は少し長い。用語 3 の説明は少し長い。用語 3 の説明は少し長い。用語 3 の説明は少し長い。用語 3 の説明は少し長い。

実際に用いる場合は、以下のようにプリアンブルでコマンドを定義しておくのが楽です。もし複数段落にまたがる説明を書く予定がないときは、以下のようなコマンドを定義しておけば十分でしょう。

```
let-block +dd dt it-dd = '< +ditem?:(fun it -> {\emph{#it;}}){#dt;}{#it-dd;}<> >
```

上のように定義された +dd コマンドを用いれば、+ditem をそのまま用いるよりも楽に定義リストを記述できます。

```
+description<
  +dd{用語 1}{ここに用語 1 の説明が入る。}
  +dd{用語 2}{
    ここに用語 2 の説明が入る。用語 2 の説明は少し長い。
    用語 2 の説明は少し長い。用語 2 の説明は少し長い。用語 2 の説明は少し長い。
  }
  +dd{ちょっと長めの用語 3}{
    ここに用語 3 の説明が入る。用語 3 の説明は少し長い。
    用語 3 の説明は少し長い。用語 3 の説明は少し長い。用語 3 の説明は少し長い。
  }
>
```

用語 *1*　ここに用語 1 の説明が入る。

用語 *2*　ここに用語 2 の説明が入る。用語 2 の説明は少し長い。用語 2 の説明は少し長い。用語 2 の説明は少し長い。用語 2 の説明は少し長い。

ちょっと長めの用語 *3*　ここに用語 3 の説明が入る。用語 3 の説明は少し長い。用語 3 の説明は少し長い。用語 3 の説明は少し長い。用語 3 の説明は少し長い。

5. 機能一覧

5.1. Enumitem モジュール

Enumitem モジュールは本パッケージの根幹となるコマンドを定義します。最も大切なのは

+item コマンドであり、Enumitem パッケージで提供される主要なコマンドは全て +item を用いたブロックボックス列に展開されるようになっています。

+item の引数は以下のようになっています。

```
+item?:( itemfmt-self )({ text-body })?:( itemfmt-child )< children >
```

itemfmt-self (itemfmt、オプション)

自分自身の項目の体裁。省略した場合は、自身の親で設定されたフォーマットが用いられる。あくまで自分自身の項目のみに影響があり、後述の通り、自身の子要素には反映されない。

text-body (inline-text、必須)

自分自身の項目の本文。ラベルは本文の左端に付く。空のインラインテキストを指定することもできる。その場合はフォーマットの display-label-with-empty-body オプションによってラベルが付くかどうか変化する。

itemfmt-child (itemfmt、オプション)

子要素の項目の体裁のデフォルト値。省略した場合、子要素は自身の親で設定されたフォーマットを引き継ぐ（itemfmt-self の値は用いられない）。

5.2. itemfmt 型

itemfmt 型は enumitem パッケージで定義されている最も重要な型であり、箇条書きの体裁の全てを決定づけるものです。itemfmt 型は context -> int list -> itemconfig 型のエイリアスであり、現在のテキスト処理文脈と、対象となる項目のインデックスの列を与えたときにラベルの体裁やインデント量といった設定値をまとめたレコードを返すような関数です。itemconfig は以下のフィールドを持つレコードとなっています。

indent (length -> length)

基準となるインデント量を与える関数。より具体的には、「label フィールドの値で決定されるラベルのボックスの横幅を与え、インデント量を返す関数」。項目は先頭の行を除き、全てここで決定される長さだけインデントされる。

label (inline-boxes)

ラベルを表すインラインボックス列。

label-with-empty-body (labelwithemptybody)

項目の中身（項目の直後に続くインラインテキスト）が空だったとき、その項目をどの

ように描くか。labelwithemptybody は以下の 3 つの定数ヴァリアントからなる代数的データ型である。

LabelDisplay　テキストが空でも構わずにラベルを描画する。

LabelInherit　自身はラベルを描画せず、自身の空でない子項目があればそこで一緒に描画させてもらう。

LabelIgnore　ラベルを描かず、継承もしない。

margin-top　(length)

箇条書きの段落の上に追加であける余白。ただし、SATYSFI 標準の機能で設けられる段落間空白はこれと独立に挿入される。

margin-bottom　(length)

箇条書きの段落の下に追加であける余白。ただし、SATYSFI 標準の機能で設けられる段落間空白はこれと独立に挿入される。

context-body　(context)

テキスト処理文脈。ラベル以外の全ての本文が与えられた関数によって変換される。