# enumitem パッケージ

monaqa

2019/09/07

## 目次



## 1　enumitem パッケージの概要

enumitem は，組版用言語 SATYSFI において，豊富な箇条書きリストや番号付きのリストを出力するためのパッケージです．SATYSFI に標準で用意されているパッケージとして itemize がありますが，enumitem パッケージは標準よりも豊富な機能を提供します．具体的には以下のような機能です：

- デフォルトで選べる豊富なスタイル
- ユーザーによる容易なスタイル拡張
- ネストの可能な番号付き箇条書き環境（2019 年 9 月現在，標準ではサポートされていません）

以下では，これらの特徴について具体的に説明します．

### 1.1　デフォルトで選べる豊富なスタイル

enumitem パッケージでは，`+genlisting` というコマンドを提供します．これは例えば以下のように使用します．

```
+genlisting(label-arabic-dot){
```

```
  * hoge
    ** hoge1
    ** hoge2
  * fuga
  * piyo
}
```

すると，以下のような結果が得られます．結果から分かる通り，箇条書きのネストもサポートしています．

1. hoge
    1. hoge1
    2. hoge2
2. fuga
3. piyo

`+genlisting` コマンドの第一引数 `label-arabic-dot` は「アラビア数字にドットを付けたラベルを用いる」ということを表しています．ここを適切に変更することで，以下のように自由自在にラベルを変更することができます．たとえばここを `label-roman-paren` とすれば以下のようなリストになります．

(i) hoge
    (i) hoge1
    (ii) hoge2
(ii) fuga
(iii) piyo

2019/09/08 現在，`arabic`, `roman`, `Roman`, `alph`, `Alph` の 5 種類の記号をインデックス部分として選択することができ，また，`raw`, `dot`, `paren`, `bracket`, の 4 種類を装飾部分として選択することができます．

番号のつかない箇条書きも同様に `+genlisting` コマンドで生成することができます．以下は `label-bullet` を第一引数としたときの箇条書きの例です．

- hoge
    - hoge1

    - hoge2
- fuga
- piyo

実用的な文書において，何種類ものスタイルを使い分けなければならない機会というのは稀かもしれません．文書中には一種類しか箇条書きが出現しないにもかかわらず，箇条書きを書くたびにいちいちスタイルを指定しなければならないのは面倒です．幸い，SATYSF$_{\mathrm{I}}$ では LaTeX と同じように自分の好きなコマンドを定義することができます．もし `+mylisting` が `+genlisting(label-bullet)` と同様に動作するようにコマンドを定義したければ，プリアンブル部分に以下のように書くことで実現できます：

```
let-block +mylisting item = '<
  +genlisting(label-bullet)(item);
>
```

なお，enumitem パッケージにおけるデフォルトの `+listing` コマンドも，やはり上の方法で `+genlisting` コマンドから定義されています．ラベル指定は `label-bullet` であり，以下のようなスタイルになっています．

- hoge
    - hoge1
    - hoge2
- fuga
- piyo

# 2　コマンドとオプション一覧

ここで紹介するのは，いわばインターフェースです．

## 2.1　コマンド一覧

enumitem パッケージでは，箇条書きを作成するためのコマンドとして以下のコマンドが定義されています．

2019 年 10 月 27 日現在，LaTeX でいうところの description 環境に相当するコマンドはまだ実装されていません．

## 2.2　インデックス指定パラメータ

インデックスを指定するためのパラメータです．以下の種類があります．

これらの正体が何であるか，気になる方もいると思います．+genlisting の第 1 引数に指定する列挙型のごとく振る舞っていますが，その実体は「現在のテキスト処理文脈とインデックス値を受け取り，ラベルとすべきインラインボックス列を返す」役割を持った context -> int -> inline-boxes 型の関数です．後述するようなカスタマイズを行わない限りユーザがこの関数を再定義したり明示的に引数を渡したりする機会はないため，本文中では列挙型と同じように扱うことができます．逆に言えば，ユーザ側が context -> int -> inline-boxes 型の関数を定義したならば，それがそのままユーザ定義された新たなインデックス指定パラメータとなります．

# 3　カスタマイズ

ここでは，enumitem パッケージを用いて自身の好みの箇条書き環境を作成するにはどのようにすればよいかについて述べます．内部の実装や型についても触れます．

## 3.1　単純な実装：数字の装飾を変える

前章のコマンド一覧で紹介したとおり，enumitem パッケージではすでに 20 以上のインデックス指定関数が定義されています．しかし，それらはそれぞれ独立に定義されているわけではなく，効率的にコードを使い回すように定義しています．現に番号付きのインデックス指定関数を見れば，それらは実際には 2 つの要素の掛け合わせで構成されている（数学的には直積）ことが容易にわかるでしょう．具体的には，たとえば label-arabic-raw 関数は以下のように定義されています：

```
let label-arabic-raw = Enumitem.label-raw Enumitem.to-arabic
```

ここで Enumitem.label-raw は (int -> inline-text) -> context -> int -> inline-boxes の型を持つ関数であり，一方で Enumitem.to-arabic は int -> inline-text の型を持つ関数です．

enumitem パッケージ内では，このように 5 種類の to- 型関数（ Enumitem.to-arabic, Enumitem.to-roman, Enumitem.to-Roman, Enumitem.to-alph, Enumitem.to-Alph ）と 4 種類の label- 型関数（ Enumitem.label-raw, Enumitem.label-dot, Enumitem.label-paren, Enumitem.label-bracket ）を定義することにより，20 種類のインデックス指定関数を生み出しています．

この性質をうまく使えば，ユーザ定義の箇条書き環境をかんたんに定義することができます．たとえば，「**問 1.**」のようなインデックスを持つ箇条書き環境を定義してみま

しょう. 1 はアラビア数字ですから, `Enumitem.to-arabic` 関数をそのまま用いることができます.

**問 1.** こんな感じ.

**問 2.** こんなふうにできる.