Enumitem パッケージ

@monaqa

目次

1. 本パッケージの概要

Enumitem は，組版システム 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 には既に itemize というパッケージが標準で用意されていますが，本パッケージではより豊富な機能を提供します（2020 年 5 月 23 日現在）．具体的には，本パッケージを使うことで以下のような恩恵を受けられます．

- デフォルトで豊富なスタイルを選択できる
- 番号付き箇条書き環境をネストさせることができる[*1]
- 定義リストを作成できる
- ネストごとに箇条書きのスタイルを変更できる
- ユーザ自身がスタイルを容易に拡張できる

なお本ドキュメントは，enumitem パッケージ (v2.0.0) の仕様および使い方について述べたものです．本パッケージは v1.x.x から v2.0.0 にバージョンアップする際にほぼ全てのコマンドの仕様を変更しており，v1.x.x を使っている場合はこのドキュメントに書かれている内容では動きません．旧バージョンの enumitem パッケージを用いたい場合は旧版ドキュメントを確認してください．

2. 基本的な使い方

パッケージを使うには当然ながら読み込みを行う必要があります．パッケージのソースファイルが SATYSFI の処理系の認識可能な位置に置かれていれば，文書の冒頭に以下の 1 行を追加するだけで読み込むことができます．本書では，以下常に `enumitem` パッケージが読み込まれているものとして説明を行います．

```
@require: enumitem
```

本パッケージは標準のものと同様，`+listing` 及び `+enumerate` という箇条書きインターフェースを提供します．デフォルトでは以下のように箇条書きを書くことができます．

```
+listing{
  * hoge
```

1 2020 年 5 月現在，標準ではサポートされていません．

```
  * 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

このように番号つき箇条書き環境のネストもサポートしており，ネストの深さによってラベルの体裁を変えることができます．

また，文章の途中で箇条書きをはさむ場合は \listing や \enumerate コマンドを使うことで改段落を伴わずに箇条書きを挿入できます．

```
+p{
  寿限無，寿限無，五劫のすりきれ，
  海砂利水魚の
  \listing{
    * 水行末
    * 雲来末
    * 風来末
  }
  食う寝るところに住むところ ...
}
```

寿限無，寿限無，五劫のすりきれ，海砂利水魚の

- 水行末
- 雲来末
- 風来末

食う寝るところに住むところ ...

ここまでの使用法は標準パッケージで提供されるものと大きく変わりません．

3. ラベルの体裁の指定

本パッケージが提供する +listing や +enumerate といったコマンドは，箇条書きのラベル（項目の先頭につく数字や記号）の体裁を指定されたものに変更する機能も持っています．

3.1. +listing のオプション引数を用いた指定

+listing 及び +enumerate は以下のように 1 つのオプション引数を受け付け，ラベルの体裁を指定することができます．

```
+listing?:(Enumitem.white-bullet){
```

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

- hoge
  - hoge1
  - hoge2
- fuga
- piyo

```
+enumerate?:(Enumitem.dot-arabic){
  * hoge
    ** hoge1
    ** hoge2
  * fuga
  * piyo
```

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

上の例における `white-bullet` や `dot-arabic` は `Enumitem` と呼ばれるモジュール内で定義されており，`labelfmt` 型を持ちます．モジュール内で定義されている `labelfmt` 型の値は他にも多数存在し，それらをオプション引数に渡すことで箇条書きを模様替えできるという仕組みです．

なお，実際のマークアップで変数名の前に `Enumitem.` を付けるのは多少面倒と感じられ

るかもしれません. Enumitem モジュールを open すれば，それ以降モジュール名を省略することができます.

```
@require: enumitem
open Enumitem
in
document(| 中略 |)'<
  +listing?:(white-bullet){
    * hoge
    * fuga
  }
>
```

このように書くと，Enumitem モジュール内で定義された値や関数の前に Enumitem. を付ける必要がなくなります．その代わり，他のパッケージやユーザで定義した関数名と衝突しないようにするのは使用者の責任となります.

また，箇条書きを書くときにいちいち同じオプション引数を付けるのは面倒だと感じられるかもしれません． その場合は最初に +listing のデフォルトの挙動を変更してもよいでしょう． プリアンブル部分に以下のように書くと， 以下のコードでは +listing を用いた場合は常に白丸がラベルとして使われるようになります.

```
let-block +listing item = '<
  +Enumitem.listing?:(Enumitem.white-bullet)(item);
>
```

余談ですが，+listing と +enumerate の違いはオプション引数を省略したときのデフォルトの体裁のみであり， オプション引数を設定すると内部での処理は全く同一となります. したがって，+listing?:(Enumitem.dot-arabic) のように +listing で番号付き箇条書きを書くことも，+enumerate?:(Enumitem.bullet) のように +enumerate で番号のない箇条書きを書くことも可能ではあります[*2].

3.2. ラベルの体裁を定める変数一覧

以下は Enumitem モジュール内で定義されている体裁指定用変数の一覧です．番号付きの箇

2 もちろん仕組み上そうなっているというだけであり，番号のない箇条書きには +listing を，番号付きには +enumerate を，などと使い分けたほうが可読性は向上するでしょう.

条書きを行いたい場合は，以下の変数を使用できます．

- アラビア数字系
  - 1 raw-arabic
  - 2. dot-arabic
  - (3) paren-arabic
  - [4] bracket-arabic
- ローマ数字系
  - i raw-roman
  - II raw-Roman
  - iii. dot-roman
  - IV. dot-Roman
  - (v) paren-roman
  - (VI) paren-Roman
  - [vii] bracket-roman
  - [VIII] bracket-Roman
- アルファベット系
  - a raw-alph
  - B raw-Alph
  - c. dot-alph
  - D. dot-Alph
  - (e) paren-alph
  - (F) paren-Alph
  - [g] bracket-alph
  - [H] bracket-Alph

また，番号の無い箇条書きには以下の変数を使用できます．

- bullet
- white-bullet

3.3. ラベルの体裁をユーザ定義する方法

ここまでの説明で，「dot-arabic 等は labelfmt 型の関数だ」 と述べましたが， そもそも labelfmt 型とは何でしょうか．実は labelfmt 型とは int list -> context -> inline-boxes という型のシノニムであり，

- 箇条書きのラベルのリスト (int list)
- 現在の本文のテキスト処理文脈 (context)

を入力として， 箇条書きのラベルを描画するインラインボックス列 (inline-boxes) を返す関数を表します．これは必ずしも予めモジュール内で用意された関数である必要はありません．つまり，labelfmt 型を持つ関数を自作して +listing コマンドのオプション引数に渡せば，自分の定義したラベル表示を持つ箇条書きを作成することができます．

labelfmt の使い方を説明するため，まず labelfmt 型を持つ比較的単純な関数を定義してみましょう．下のコードを動かすには，標準で提供されている list パッケージを読み込む必要があります．

```
let example idxlst ctx =
  let idx-to-ib idx =
    (read-inline ctx (embed-string (arabic idx)))
      ++ (read-inline ctx {.})
  in
  idxlst |> List.map idx-to-ib
         |> List.fold-left (++) inline-nil
```

この関数は labelfmt (int list -> context -> inline-boxes) 型を持ちます．まず， 関数内部で定義された idx-to-ib は int -> inline-boxes 型を持つクロージャであり， この関数を idxlst の要素のそれぞれに適用して inline-boxes list 型へと変換します．続いて得られたリストを左から結合することにより， 最終的な inline-boxes 型の値を得ます．したがって，もし example の第 1 引数に [2; 3; 5; 7] というリストが与えられた場合，（第 2 引数のテキスト処理文脈により組み方は変わりますが）「2.3.5.7.」という数字の列を持ったインラインボックス列が出力となります．

では， この example を +listing に指定するとどうなるでしょうか．実際に組んでみると以下のようになります．

```
+listing?:(example){
```

```
  * あああ
  * いいい
  * ううう
    ** あああああ
    ** いいいいい
    ** ううううう
  * えええ
    ** あああああ
      *** あああああああ
      *** いいいいいい
      *** うううううう
    ** いいいいい
  * おおお
}
```

1.あああ
2.いいい
3.ううう
    1.3.あああああ
    2.3.いいいいい
    3.3.ううううう
4.えええ
    1.4.あああああ
        1.1.4.あああああああ
        2.1.4.いいいいいい
        3.1.4.うううううう
    2.4.いいいいい
5.おおお

結果を見れば規則性が分かると思いますが，+listing コマンドは labelfmt 型の関数が与えられたとき，labelfmt に「現在の箇条書きの項目が何番目か」という情報（インデックス）を格納したリストと現在のテキスト処理文脈の 2 つを渡し，出てきた出力をそのまま箇条

書きのラベルとします．なお，与えられるインデックスのリストでは n 番目に深いリストのインデックスが「末尾から数えて n 番目の要素」に格納されています．つまりリストの先頭にあるものが最も深い，末尾にあるものが最も浅いネストの番号となります．これは少し直感に反するかもしれませんが，逆に並べるより実装が楽になることが多いためこのような仕様にしました．

この仕様さえ理解すれば，好きな箇条書き環境を定義できます．たとえば，「問 1.」のようなラベルは以下のようにして実装できます．

```
let label-toi idxlst ctx =
  let idx = match idxlst with
    | [] -> 1
    | idx :: _ -> idx
  in
  let it-num = embed-string (arabic idx) in
  read-inline ctx {問 #it-num;.\ }
```

let idx = ... から始まる行で idxlst の一番最初の要素を取り出し，それをインラインテキストに変換しています．実際には enumitem パッケージの内部で label-toi の引数に空リストが与えられることはないはずであり，それを踏まえればもう少し簡潔に書くこともできます[*3]．

```
let label-toi (idx :: _) ctx =
  let it-num = embed-string (arabic idx) in
  read-inline ctx {問 #it-num;.\ }
```

話が少しそれました．実際に label-toi を使ってみましょう．

```
+enumerate?:(label-toi){
  * hoge
  * fuga
    ** fuga1
      *** fuga11
      *** fuga12
```

3 ただし，ユーザが故意に与えた，本パッケージに不具合があったなどの理由で label-toi の引数に空リストが与えられてしまった場合は実行時エラーとなるため，その点には注意が必要です．

```
    ** fuga2
}
```

問 1. hoge

問 2. fuga

　問 1. fuga1

　　問 1. fuga11

　　問 2. fuga12

　問 2. fuga2

このようにネストがある場合でも，ネストに応じて適宜リセットされながらラベルが表示されることが分かります．では，もし以下のように定義したらどうなるでしょうか．

```
let label-toi-nest idxlst ctx =
  let str-nums =
    idxlst |> List.reverse
           |> List.map (fun idx -> arabic idx ^ `.`)
           |> List.fold-left (^) ` `
  in
  let it-num = embed-string str-nums in
  read-inline ctx {問 #it-num;\ }
```

ある程度予想がついたかもしれませんが，ラベルもネストされた状態で表示されます．

```
+enumerate?:(label-toi-nest){
  * hoge
  * fuga
    ** fuga1
      *** fuga11
      *** fuga12
    ** fuga2
}
```

問 1. hoge

問 2. fuga

　問 2.1. fuga1

　　問 2.1.1. fuga11

　　問 2.1.2. fuga12

　問 2.2. fuga2

なお，第 1 引数を関数内で使わなければ，番号のない箇条書きの体裁も作成できます．

```
let label-japanese-ichi _ ctx =
  read-inline ctx {一} ++ inline-skip 10pt
```

番号つき箇条書きよりも更にシンプルな定義になりました．このように定義された箇条書きは以下のように用いることができます．

```
+listing?:(label-japanese-ichi){
  * 廣ク會議ヲ興シ萬機公論ニ決スベシ
  * 上下心ヲ一ニシテ盛ニ經綸ヲ行フベシ
  * 官武一途庶民ニ至ル迄各其志ヲ遂ケ人心ヲシテ倦マサラシメン事ヲ要ス
  * 舊來ノ陋習ヲ破リ天地ノ公道ニ基クベシ
  * 智識ヲ世界ニ求メ大ニ皇基ヲ振起スベシ
}
```

一　廣ク會議ヲ興シ萬機公論ニ決スベシ

一　上下心ヲ一ニシテ盛ニ經綸ヲ行フベシ

一　官武一途庶民ニ至ル迄各其志ヲ遂ケ人心ヲシテ倦マサラシメン事ヲ要ス

一　舊來ノ陋習ヲ破リ天地ノ公道ニ基クベシ

一　智識ヲ世界ニ求メ大ニ皇基ヲ振起スベシ

3.4. 便利な関数

ネストの深さに応じて箇条書きのラベルを変えたい，という要求は珍しくありません．ラベルの体裁を自身で定義すれば当然そういった箇条書きも作成できます．しかし，既存の体裁

を使いまわしたいのであれば，Enumitem.change-by-depth 関数を用いるのがより簡単です．

```
+listing?:(
  Enumitem.change-by-depth [
    Enumitem.bracket-Alph; Enumitem.white-bullet;
    Enumitem.paren-arabic;
  ]
){
  * hoge
  * fuga
    ** fuga1
      *** fuga11
      *** fuga12
    ** fuga2
}
```

[A] hoge

[B] fuga

- fuga1
  (1) fuga11
  (2) fuga12
- fuga2

4. 本文の体裁の指定

前節でラベルの体裁の変更方法を説明しましたが，+listing コマンドで指定できるのはラベルの体裁のみであり，本文のフォントや本文が複数行に渡るときのインデント量といった，本文に関係する体裁を指定することはできませんでした．そういった要素にも変更を加えたい場合は，箇条書きコマンド +listing や +enumerate をより一般化した +genlisting コマンドを使用します．+genlisting は +listing にくらべて複雑なインターフェースを持

ちますが，その分自由度が高く

- 本文のフォントサイズや色といった体裁
- 箇条書きの項目間の幅
- 1 つの項目内で行を折り返すときのインデント幅
- ネストするときに追加されるインデントの幅

といった項目を全てこの関数 1 つで制御することができます．

4.1. +genlisting コマンドの仕様

実は，+listing や +enumerate コマンドも +genlisting を用いて定義されています．まずは +listing コマンドがパッケージ内部でどのように定義されているか見てみましょう．

```
let-block ctx +listing ?:labelf item =
  let labelf = Option.from bullet labelf in
  read-block ctx '<+genlisting(labelf)(default-item)(item);>
```

このように，+genlisting は +genlisting(labelf)(itemf)(item); という形で用いられます（オプション引数はありません）．個々の引数の意味は以下のとおりです．

labelf (labelfmt (int list -> context -> inline-boxes))
: ラベルの体裁を指定する関数．

itemf (context -> int -> inline-boxes -> inline-text -> block-boxes)
: ラベル及び本文の組み方を指定する関数．以下の引数を与えると，箇条書きの項目を組んでブロックボックス列を返す：

  1. 箇条書きが組まれる位置でのテキスト処理文脈 (context)，
  2. 対象となる項目の深さ (int)．
  3. 対象となる項目にて組まれることになるラベル (inline-boxes)．
  4. 本文を表すインラインテキスト (inline-text)

item (itemize)
: 箇条書きの内容．

1 番目は +listing のオプション引数，3 番目は +listing の必須引数と同じものですが，+genlisting では追加で 2 番目に itemf 引数を指定することができます．これは「深さとラベルと本文が与えられたときに，箇条書きの 1 つの項目を組んでブロックボックス列にする」関数であり，まさに箇条書きの体裁を直接指定する自由度の高い関数です．

+genlisting コマンドが呼び出されると，以下の処理を再帰的に行うことで箇条書きを

組みます．再帰的な処理と一項目に対する処理を分離することで，柔軟性の高い箇条書きが表現できる仕組みとなっています．

1. 第 1 引数で与えられた `labelf` を用いて親項目のラベルを組む．
2. 第 2 引数で与えられた `itemf` を用いて親項目の本文を組む（そのときに先程組んだラベルのインラインボックス列を渡す）．
3. 子項目を親と同じ要領で組み，親項目のブロックボックス列の下に結合する．

4.2. +genlisting を用いた指定

`+listing` の定義の中で `+genlisting` の第 2 引数に渡されているのは `default-item` という関数です．これは `Enumitem.default-item` とすることで使用でき，実際には以下のように定義されています：

```
let default-item ctx depth ib-label text =
  let text-indent =
    let fsize = get-font-size ctx in
    fsize *' ((EnumitemParam.get item-indent-ratio) *. (float depth))
  in
  let ib-text =
    let index-width = get-natural-width ib-label in
    let item-text-width =
      (get-text-width ctx) -' text-indent -' index-width
    in
    embed-block-top ctx item-text-width (fun ctx ->
      line-break true true ctx (read-inline ctx text ++ inline-fil)
    )
  in
  line-break true true ctx
    ((inline-skip text-indent) ++ ib-label ++ ib-text)
```

この関数には，ラベルを受け取って本文を組み上げる処理の本質が書かれています．基本的にはこの関数をいろいろいじって自分好みの関数に仕上げ，それを `+genlisting` の第 2 引数にわたすことによって，箇条書きの体裁を自由に変更することができます．しかし，本文のテキスト処理文脈をいじる程度の変更であれば，`default-item` をそのまま用いて比較的簡単に実装することができます．

```
% プリアンブルに書く
let gray-item ctx =
  let ctx2 = ctx |> set-text-color (Color.gray 0.7) in
  Enumitem.default-item ctx2

% 本文中に書く
+genlisting(Enumitem.paren-arabic)(gray-item){
  * hoge
  * fuga
    ** fuga1
      *** fuga11
      *** fuga12
    ** fuga2
}
```

(1) hoge

(2) fuga

  (1) fuga1

    (1) fuga11

    (2) fuga12

  (2) fuga2

見ての通り，第 2 引数の関数内部でテキスト処理文脈を変更してもラベルの色は変化しません．ラベルと本文のインラインボックス列は別のテキスト処理文脈に基づいて組まれるためです．ラベルの色を変更したい場合は，前節で説明したように第 1 引数を変更します．

```
% プリアンブルに書く
let paren-arabic-gray idxlst ctx =
  let ctx2 = ctx |> set-text-color (Color.gray 0.7) in
  Enumitem.paren-arabic idxlst ctx2

% 本文中に書く
```

```
+genlisting(paren-arabic-gray)(gray-item){
  * hoge
  * fuga
    ** fuga1
      *** fuga11
      *** fuga12
    ** fuga2
}
```

(1) hoge

(2) fuga

    (1) fuga1

        (1) fuga11

        (2) fuga12

    (2) fuga2

5. グローバルパラメータを用いた指定

本パッケージで制御できる箇条書きの体裁は，ほぼ全て今までに述べてきた方法でカスタマイズすることができます．とはいえ，ユーザが何かを変更したいと思うたびに，それに応じて新たな関数を用意しなければならないのは面倒かもしれません．たとえば，箇条書きの字下げ幅は本来 1 自由度しかありませんから 1 つのパラメータの値を変更すれば対処できそうなものですが，今までの方法では「箇条書きの字下げ幅を変更した本文体裁指定用関数」を新たに定義する必要があります．その面倒さを避けるため，本パッケージではいくつかの設定項目を「グローバルパラメータ」として保持しており，必要に応じて狙い撃ちして変更することができます．

5.1. グローバルパラメータの種類

現時点では，以下のグローバルパラメータが Enumitem モジュールに用意されています．

item-indent-ratio　float 型のパラメータ (default: 2.0)

現在のフォントサイズに対する相対的な割合を指定して，箇条書きのネストが行われたときに，自分の親の項目から行うインデントの量を表す．

label-width-ratio　float 型のパラメータ (default: 2.0)

現在のフォントサイズに対する相対的な割合を指定して，デフォルトで定義されている `raw-arabic` や `bullet` などのラベルの幅を表す．これらのラベルは右揃えで組まれるが，その際に必ず (`font-size *' label-width-ratio`) の分だけの空白が左端から確保される．

「float 型のパラメータ」と便宜上書いているものは，正確には `EnumitemParam` モジュールにて定義された `float EnumitemParam.t` という型です． パラメータを指定したりデフォルト値を設定したりするインターフェースが用意されているものの，基本的には `float ref` と似たような型である，という理解で問題ありません．

なお，新たなグローバルパラメータをユーザ定義することもできますが，箇条書きという趣旨から外れるためここでは割愛します[*4]．

5.2. パラメータの値の変更

これらのパラメータに好きな値を設定する方法はいくつかあります．1 つ目は，プリアンブル部分で以下のように指定する方法です．文書全体に共通する値を指定するときに使います．

```
let () = Enumitem.item-indent-ratio |> EnumitemParam.set 5.0
```

このように書くと， 本文中では常に `item-indent-ratio` の値が 5.0 であるものとして組まれます．`EnumitemParam.set` は `'a -> 'a EnumitemParam.t -> unit` 型を持つ関数で，指定したパラメータに特定の値を代入することができます．

2 つ目は， 本文中で `+set-param` や `\set-param` コマンドを用いる方法です． この方法を用いると，本文の途中であってもパラメータを変更することができます．

```
+listing{
  * hogehoge
  * fugafuga
}
+set-param(Enumitem.label-width-ratio)(4.0);
```

4 気になる場合は，本パッケージのソースを読んでいただければなんとなく使い方が分かるかと思います．

```
+p{ここからラベルの幅が広くなる．}
+listing{
  * hogehoge
  * fugafuga
}
+p{ここからラベルの幅が狭くなる．
   \set-param(Enumitem.label-width-ratio)(1.0);}
+listing{
  * hogehoge
  * fugafuga
}
```

- hogehoge
- fugafuga

ここからラベルの幅が広くなる．

- hogehoge
- fugafuga

ここからラベルの幅が狭くなる．

- hogehoge
- fugafuga

しかし，パラメータを途中から永続的に変えるのは，後ろの原稿に思わぬ影響を与えうるという点で多少リスキーでもあります．そこで，3 つ目の方法として一時的にパラメータの値を変更するインターフェースも用意されています．以下の `+with-param` や `\with-param` コマンドを用います．

```
+listing{
  * hogehoge
  * fugafuga
}
+with-param(Enumitem.label-width-ratio)(4.0)<
  +p{ここだけラベルの幅が広くなる．}
```

```
  +listing{
    * hogehoge
    * fugafuga
  }
>
+p{ここはラベルの幅が前と同じ．}
+listing{
  * hogehoge
  * fugafuga
}
```

- hogehoge
- fugafuga

ここだけラベルの幅が広くなる．

  - hogehoge
  - fugafuga

ここはラベルの幅が前と同じ．

- hogehoge
- fugafuga

6. 動的なフラグを用いたラベル操作

6.1. +xgenlisting コマンド

たとえば，以下のような To-Do リストを作成したくなったとします．

□ 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.

簡単のため，ラベルを作成するための関数は既に用意されているものとしましょう．たとえば，以下の関数 square-label は bool -> context -> inline-boxes 型を持ち，第 1 引数 is-checked が true のときチェック済みのチェックボックスを，false のときチェック済みでないチェックボックスを描画します.

```
let square-label is-checked ctx =
  let fs value = (get-font-size ctx) *' value in
  let fsize = fs 1.0 in
  let gr-square (x, y) =
    stroke 0.5pt Color.black
      (Gr.rectangle (0pt, 0pt) (fs 0.5, fs 0.5 ))
      |> shift-graphics (x, y +' fs 0.15)
  in
  let gr-mark-done (x, y) =
    stroke 0.5pt Color.black (
      Gr.poly-line (fs (0.0 -. 0.1), fs 0.45)
        [(fs 0.15, fs 0.15); (fs 0.75, fs 0.85)])
        |> shift-graphics (x, y +' fs 0.15)
  in
  let gr point =
    if is-checked then
      [gr-square point; gr-mark-done point]
    else
      [gr-square point]
  in
  inline-skip 10pt ++ (inline-graphics fsize fsize 0pt gr)
```

今までに紹介された +listing を用いても，上のように「3 番目のみチェックが付いたリスト」を実現することはできます．以下のようにパターンマッチなどを用いて，チェックを付けたい場所のみ場合分けして処理すればよいのです.

```
+listing?:(fun idxlst ctx -> (
```

```
  let is-checked = match idxlst with
    | [3] -> true
    | _ -> false
  in
  square-label is-checked ctx
)){
  * ミルクを買う.
  * The \SATySFi;book を読む.
  * \SATySFi; を完全に理解する.
  * 課題を解く.
}
```

しかしこれは今回の用途を考えるとあまり直観的ではなく，また編集もしやすいとはいえません．「ミルクを買う」にチェックを付けたいときに本文から離れた場所を編集しなければならないのは手間ですし，「どこにチェックが付いているのか」をひと目で判断することができません．また今回は項目数が 4 つでしたが，もっと長いリストでいちいち上からインデックスを数えるのは手間です．ラベル指定用の関数は，このように例外的にまたは動的にレイアウトを変更する用途にはあまり適していません．

本パッケージではこのようなケースに対応するため，本文中でパラメータに値をセットし，その値をラベルに反映させる方法を提供しています．たとえば先程の To-Do リストは，プリアンブルにて以下のように定義された `+todo-list` 及び `\done` コマンドで作成できます．

```
let done = EnumitemBase.make-param false
let-inline \done = {\set-item(done)(true);}
let-block +todo-list item = '<
  +xgenlisting
    (fun idx ctx -> square-label (EnumitemParam.get done) ctx)
      (default-item)(item);
>
```

この `+todo-list` 及び `\done` コマンドを使うと，先程の箇条書きは以下のようにシンプルに書くことができます．

```
+todo-list{
  * ミルクを買う.
```

```
  * The \SATySFi;book を読む.
  * \done; \SATySFi; を完全に理解する.
  * 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.

このように，本文に \done; というコマンドが付いているアイテムに限りチェックマークが付くようになります．\done の位置は，3 番目のアイテムの本文中であればどこにあってもかまいません．

何が起きたのか，もうすこし詳細に説明します．ポイントは以下の 4 点です．

- EnumitemBase というモジュールの make-param 関数を用いて，箇条書きのアイテム内で一時的な値を保持するための器（ローカルパラメータ）を定義することができる．
- ローカルパラメータには，本文中で \set-item(param)(value); と打つことで一時的に値を設定することができる．
- +xgenlisting を使うと，ラベルの体裁を指定する際に EnumitemParam.get 関数でパラメータの値を受け取り，その値に応じてラベルの出力を変更することができる．
- \set-item で変更した値はその項目が終わると破棄され，デフォルト値にもどる．

先程の例の最初にある EnumitemBase.make-param という関数で，done という名前のローカルパラメータを生成しています．このローカルパラメータは前節で説明したグローバルパラメータとは少し異なり，箇条書きの項目の内部でパラメータを変更しても，その項目が終わったときに設定が破棄されます．したがって 3 番目の項目に限って \done コマンドをつけた場合，3 番目のラベルにはチェックマークが付きますが，4 番目には付きません．

今回定義した done は bool 型を保持する比較的単純なローカルパラメータでしたが，それ以外にも int 型や inline-text 型など，様々な型を持つパラメータを作成することができます．工夫次第では，パラメータの値に応じてラベルのインデックスを変更したり，ラベルの体裁そのものを変えたり，と多彩なカスタマイズが可能となります．

6.2. +xgenlisting の注意点

+xgenlisting は便利なコマンドですが，+genlisting にはない欠点が存在します[*5]．それは「本文に副作用のあるコマンドを入れると不具合が起きやすい」ということです．たとえば，内部でカウンタをインクリメントさせる \footnote を +xgenlisting の本文で用いると，（実装にもよるとは思いますが）脚注の数字が 2 つインクリメントされてしまう可能性があります．

理由は実装上の事情にあります．実は，「本文中にあるコマンドを読み取り，その値に応じてラベルを更新する」という組み方は少し不自然なのです．なぜなら本来，ラベルを組み終わってはじめて後続する本文のテキスト幅が分かり，本文を組むことができるようになるからです．ラベルを組まないと本文が組めない，しかし本文を読まないとラベルを更新できない，というのが実装の困難な点でした．

そこで，+xgenlisting では「ラベルを組む前に read-inline で本文を読み，読み終わったものは一度破棄する」という方式で実装を行いました．本文中のコマンドは read-inline で読まれることにより評価され，\set-item がある場合はそのタイミングで対象のローカルパラメータが一時的な値へとセットされます．その後であればラベルを組むことができ，ラベルさえ組めれば再度改めて本文を組むことができます．

このとき，本文は read-inline によって 2 度評価されます．副作用のないコマンドであれば何度評価されても結果は変わりませんから問題ありませんが，副作用のあるコマンド，特に 2 度だけ呼ばれることを想定していないコマンドを +xgenlisting の本文のインラインテキスト中に入れると，このことによって挙動がおかしくなるのです．

7. 定義リストの作成

LaTeX や HTML において標準で用意されている箇条書きは 3 種類ありました．順序なしリスト（番号のない箇条書き），順序リスト（番号付き箇条書き），そして定義リストです．

順序なしリスト
: 順序を表すインデックスのない箇条書きです．通常，LaTeX では itemize 環境，HTML では <ul> タグにより組まれます．

順序リスト
: 順序を表すインデックスのある箇条書きです．通常，LaTeX では enumerate 環境，HTML では <ol> タグにより組まれます．

5 この欠点のため，+genlisting を +xgenlisting で置き換えることができませんでした．

定義リスト

各インデックス毎に固有の単語が付与されている箇条書きです． 通常，LaTeX では description 環境，HTML では <dl> タグにより組まれます．

この説明で用いられているリストは定義リストです．

1 番目及び 2 番目のリストは SATySFi 標準の itemize パッケージでも実現することが出来ますし，本パッケージでも実現できることは今までに述べたとおりです．それに対し，3 番目の定義リストを実現する手段は標準の itemize パッケージでは用意されていませんでした．

本パッケージでは， 定義リストを実現する +gendescription コマンドを用意しています． このコマンドを用いれば汎用性の高い定義リストを記述することができます．具体例を以下に示します．

```
+gendescription(|
  nextline = false;
  title-inner-gap = 10pt;
  inner-indent = (fun title-wid -> title-wid);
  title-func =
    (fun ctx title -> read-inline ctx {\textbf{#title;}} );
|){
  * 英語のパングラム
    ** The quick brown fox jumps over the lazy dog.

  * 日本語のパングラム
    ** いろはにほへとちりぬるを
       わかよたれそつねならむ
       うゐのおくやまけふこえて
       あさきゆめみしゑひもせすん
    ** いろはにほへとちりぬるを
       わかよたれそつねならむ
       うゐのおくやまけふこえて
       あさきゆめみしゑひもせすん
}
```

英語のパングラム　The quick brown fox jumps over the lazy dog.

日本語のパングラム	いろはにほへとちりぬるをわかよたれそつねならむうゐのおくやまけふこえてあさきゆめみしゑひもせすん いろはにほへとちりぬるをわかよたれそつねならむうゐのおくやまけふこえてあさきゆめみしゑひもせすん

具体例を見れば分かる通り，+genlisting などのコマンドとはかなりインターフェースが異なります．+gendescription は 2 つの引数 record と inner をとります．record は description コマンドの体裁を設定するレコードであり，inner は表示したい内容を並べる itemize 型の引数です．

SATYSFI には現在，LATEX の \item[label] のように箇条書きの item 毎にラベルを変更するための記法がありません．そこで，+gendescription コマンドでは inner 部分を以下のように定めることで，SATYSFI の箇条書き構文のインターフェースに沿った定義リストを実現しました[*6]．

- 1 番目の深さの item（* が 1 つ並んでいるところ）に，定義リストの各 item に相当する見出しを書く．
- 各見出しの子に相当する item（* が 2 つ並んでいるところ）に，定義リストの本文を書く．複数の子 item を同じ親の下に並べて書くことで，複数の段落を記述することができる．

また，+gendescription コマンドでは第 1 引数でレコードを渡すことにより，柔軟に定義リストを構成することができます．レコードの値は 4 種類あり，これらは全てオプションではなく必須引数です．

nextline (bool)

各 item のタイトルと本文の間に改行を挟むかどうか．true ならば改行を行う．false ならば改行を行わない．

title-inner-gap (length)

各 item のタイトルと本文の間に挿入する最小の空白幅．どちらかというと「各 item のタイトルの右に常においておく余白」というイメージが近い．

6 説明を読むよりも前述の具体例を見たほうが使用法が掴みやすいでしょう．

inner-indent (`length -> length`)

「タイトルの幅を引数として，本文インデント量を返す関数」を指定する．わざわざ関数を指定するようにしているのは，タイトル幅に応じてインデントを動的に変えられるようにするため．たとえば `(fun _ -> 10pt)` を指定すれば，本文ではタイトルの幅の長さに関わらず常に 10pt のインデントが行われる（ただし，タイトルが記述されている行を除く）．また，`(fun title-wid -> title-wid)` を指定すれば，タイトル幅（title-inner-gap を含む）と同じだけのインデントが行われる．

title-func (`context -> inline-text -> inline-boxes`)

「テキスト処理文脈及びタイトルのインラインテキストを引数として，タイトルのインラインボックス列を返す関数」を指定する．最も簡単なのは `read-inline` 関数をそのまま指定すること．この関数内部で `\textbf` のコマンドを挟んだり，テキスト処理文脈をいじってフォントを変更したりすることで，各項目のタイトルだけを太字にする，といった処理が可能になる．

なお，本パッケージには必要なレコード引数が事前に埋められた `+description` コマンドも用意されています．

```
let desc-config = (|
  nextline = true;
  title-inner-gap = 5pt;
  inner-indent = (fun title-wid -> 20pt);
  title-func = read-inline;
|)

let-block +description item = '<
  +gendescription(desc-config)(item);
>
```