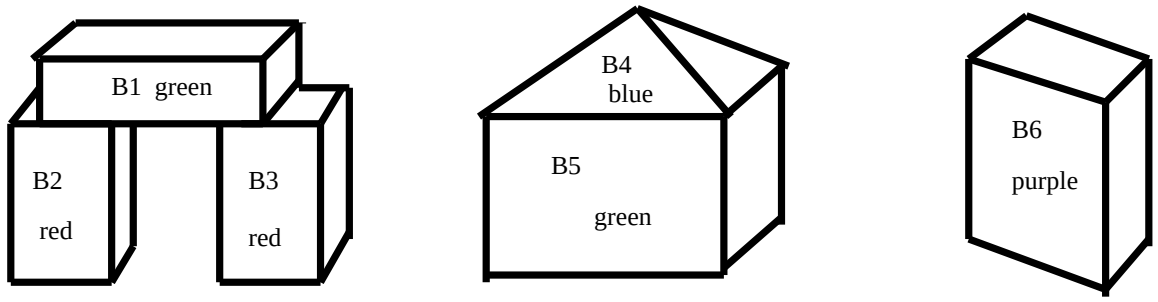
## 関数適用いろいろ (続き)

関数 every … 関数とリストを引数に取り，リストの全要素が偽とならなければ T を返す.
例:
(every #'numberp '(1 2 3 4 5)) ⇒ T
(every #'numberp '(1 a 3 b 5)) ⇒ NIL
(every #'(lambda (x) (> x 0)) '(1 2 3)) ⇒ T
(every #'(lambda (x) (> x 0)) '(1 -2 3)) ⇒ NIL
(every #'oddp nil) ⇒ T
(every #'> '(10 20 30) '(1 2 3)) ⇒ T

関数 reduce … 関数とリストを引数に取るが，関数は 2 引数を取るものだけに限る．その関数をリストの全要素に対し順次適用し，1 つの答にする.
例:
(reduce #'+ '(1 2 3)) ⇒ 6
(reduce #'+ '(1)) ⇒ 1
(reduce #'+ nil) ⇒ 0
(reduce #'append '((1 2) (3 4) (5 6))) ⇒ (1 2 3 4 5 6)

## 応用例 “積木の世界” (人工知能の古典)

下図のような積木の状況を表現してみよう.




物体の形質や状態といった知識を (ブロック名 属性名 属性値) の 3 つ組で表すことにする．すると，上図の状況は次のような 3 つ組の集まりで表現できる.

```
(b1 shape brick)
(b1 color green)
(b1 size small)
(b1 supported-by b2)
(b1 supported-by b3)
(b2 shape brick)
(b2 color red)
(b2 size small)
(b2 supports b1)
```

```
(b2 left-of b3)
(b3 shape brick)
(b3 color red)
(b3 size small)
(b3 supports b1)
(b3 right-of b2)
(b4 shape pyramid)
(b4 color blue)
(b4 size large)
(b4 supported-by b5)
(b5 shape cube)
(b5 color green)
(b5 size large)
(b5 supports b4)
(b6 shape brick)
(b6 color purple)
(b6 size large)
```

こういった知識を用いて,「ブロック b2 の色は何?」とか「ブロック b1 を支えているブロックは何?」といった質問に答える関数を作ってみよう．このような質問は，“?” を使ったパターンで表現できる:

| | | |
|---|---|---|
| 「ブロック b2 の色は何?」 | … | (b2 color ?) |
| 「ブロック b1 の色は赤?」 | … | (b1 color red) |
| 「ブロック b1 とブロック b2 の関係は?」 | … | (b1 ? b2) |
| 「ブロック b1 については何が分かっているか?」 | … | (b1 ? ?) |
| 「支えの関係にあるブロックは何か?」 | … | (? supports ?) |
| 「ブロック b1 に関係するブロックとその関係は?」 | … | (? ? b1) |
| 「この世界の全知識を知りたい!」 | … | (? ? ?) |

知識の 3 つ組集合が大域変数 `blockdata` に代入されているものとして，上のようなパターンマッチング関数を実装していこう．(`blockdata` は，UNIX コマンド
`wget http://www.nak.ics.keio.ac.jp/class/lisp/blockdata.lsp` で各自のカレントディレクトリにコピーできます．コピーしたら，gcl 上で (`load "blockdata.lsp"`) で読み込みます．)

[問 1] 2 引数に対して，それが等しいか，2 番目の引数が ‘?’ の時だけ値 T を返し，それ以外の場合は NIL を返す関数 match-element を定義しなさい.

[問 2] 2 つの 3 つ組 (知識とパターン) を引数にとり，知識がパターンに合致したら T を返す関数 match-triple を定義しなさい.
例: (match-triple '(b2 color red) '(b2 color ?)) ⇒ T
(match-triple '(b2 color red) '(b2 color blue)) ⇒ NIL
(ヒント: every と match-element を使うとよい)

[問 3] 1 つのパターンを引数に取り，そのパターンに当てはまる知識を blockdata からすべて見つける関数 fetch を定義しなさい.
例: (fetch '(? supports b1)) ⇒ ((b2 supports b1) (b3 supports b1))
(ヒント: remove-if-not と match-triple を使うとよい)

[問 4] 関数 fetch を使い，いろいろなパターンで質問しなさい.

[問 5] ブロック名を引数に取り，そのブロックの色を尋ねるパターンを返す関数 color-pattern を定義しなさい.
例: (color-pattern 'b3) ⇒ (b3 color ?)

[問 6] ブロック名を引数に取り，そのブロックを支えるブロックすべてをリストにして返す関数 supporters を定義しなさい.

[問 7] ブロック名を引数に取り，そのブロックの性質のすべてをリストにして返す関数 description を定義しなさい.
例: (description 'b2) ⇒ (shape brick color red size small supprots b1 left-of b3)
(ヒント: 段階的に定義していこう．まず引数で与えられるブロック名に関するすべての知識を収集する関数 desc1 を定義する．その結果からブロック名を取り除く関数 desc2 を定義する．そしてその結果を 1 つのリストにまとめるようにするとよい．)

[問 8] ブロック名を引数に取り，そのブロックが立方体 (cube) に支えられているかどうかを判定する述語 supp-cube を定義しなさい.
例: (supp-cube 'b4) ⇒ 真
(supp-cube 'b1) ⇒ NIL
(ヒント: 関数 supporters の返値を使って…)