Emacs

- Emacs (イーマックス) は歴史の長い代表的なテキスト・エディタである。
- コントロール・キーを使うのが由緒正しいが、とりあえず、メニューや矢印キーを使ってテキストを編集しよう。
- irb でロードする前にセーブを忘れずに。
- 自分の慣れたエディタがあれば、それを使ってもよいです。ただし、プレーン・テキストのファイルが作れないと駄目。

Emacs のコントロールキーの例

← … ^B (backward)

↑ … ^P (previous)

↓ … ^N (next)

→ … ^F (forward)

カーソルの下の文字の消去 … ^D (delete)

カーソルから行末までの消去 … ^K (kill)
（連続すると行の消去）

ペースト … ^Y (yank)

練習

- ファイル heron.rb を作り、ロードして動作を確認せよ。

進捗状況の確認

1. できた（できた時点で投票してください）
2. ファイルが作れない
3. ロードがうまくいかない
4. 関数が実行できない（エラーなど）

できてしまった人は、次のページの演習を
やってください。

練習1.9g)

- 二次方程式 $ax^2 + bx + c = 0$ に関して

  (a) 判別式 $b^2 - 4ac$ を求めるdet(a,b,c).

  (b) 解の1 つを求める solution1(a,b,c). (det を使って定義せよ。)

  (c) もう 1 つの解を求める solution2(a,b,c). (solution1 と solution2 の共通部分を1つの補助関数にできるか?)

  (d) 二次関数$f(x) = ax^2+bx+c$ の値を求める quadratic(a,b,c,x).

  各定義をquadratic.rbという名前のファイルに格納せよ。

1.6　定義のまとめ

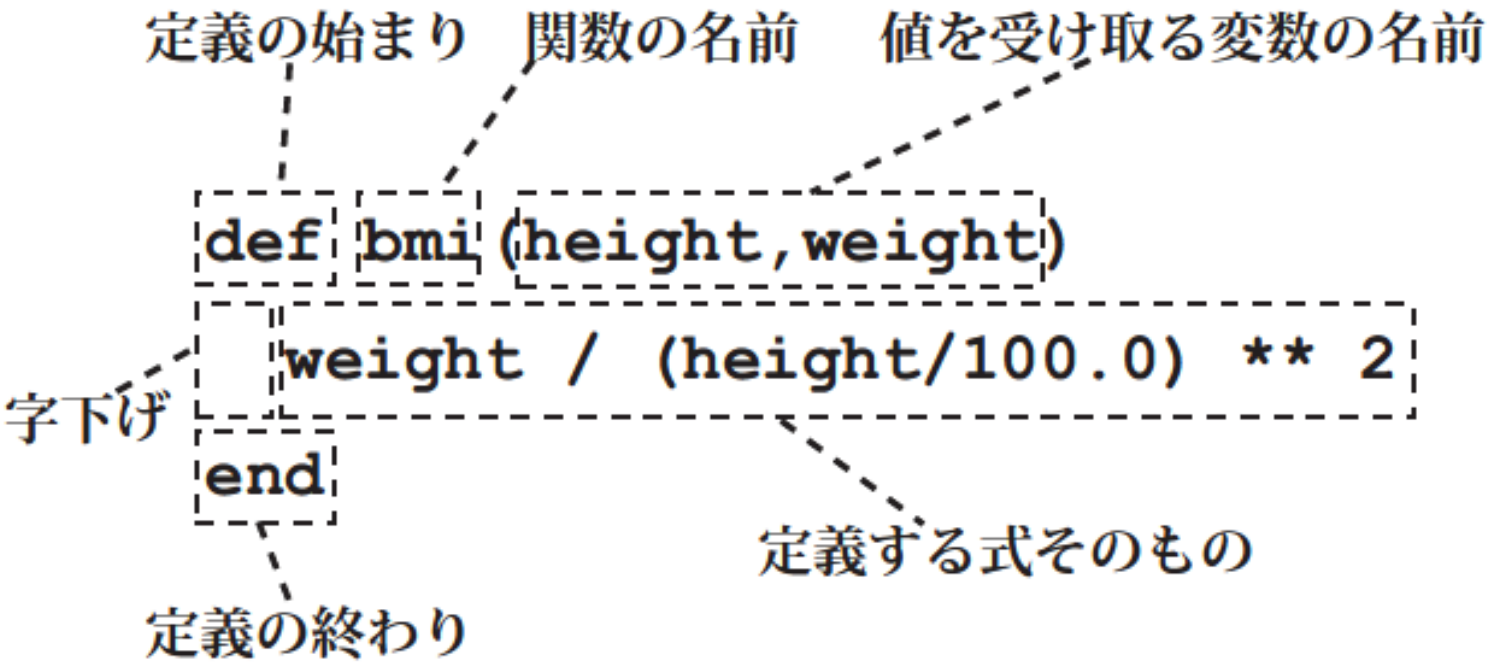
この章で紹介した式や命令をまとておこう。

`include(Math)`：　`cos` や `sqrt` などの数学関数を使う前に、実行しておかなければいけない命令。

変数名 = 式：右辺の式を計算した値を、左辺に書かれた名前の変数にしまう**代入命令**。

関数名(式$_1$,...,式$_n$)：　式$_1$,...,式$_n$を計算した値を、関数に渡す**関数呼び出し式**。関数に渡される式の値のことを**引数**(ひきすう)という。関数名は `sqrt` のような予め定義されている関数でも、`bmi` のような自分で定義した関数でもよい。

def 関数名 (変数名$_1$,...,変数名$_n$) ↵ 式 ↵ end : **関数定義**。少し複雑なので、具体例で説明する。



def と end は、定義の範囲を示している。**関数の名前**(ここでは bmi) は、変数名と同様、好きな名前を付けることができる。次の (height,weight) は、関数が値を受け取るために使う変数名である。

練習問題確認プログラム

- 練習1.3と1.9と1.10をやる
- check.rb と ex01.rb をダウンロード
- ターミナルの中で以下のように実行する

  $ ruby check.rb ex01.rb
- うまくできたことを確かめたら以下を実行

  $ ruby check.rb --report ex01.rb
  - 結果、ex01-091021-204855.tgz というようなファイルが作られる
  - これをメールに添付して送る
  - メールの本体には学生証番号と名前を忘れずに

練習問題確認プログラム

- ex01.rb（練習1.3と1.9と1.10）の結果を以下のアドレスに送れ。

  is-komaba@lyon.is.s.u-tokyo.ac.jp

- Subject:欄は、

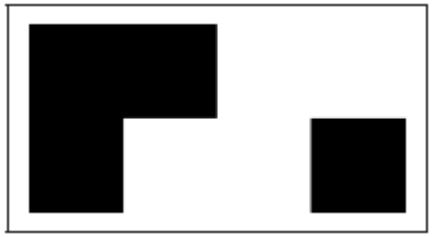
  is-komaba

  としてください。

- 〆切: 11月1日
  - 練習問題確認プログラムの結果は出席点の一部とします。結果の内容は問いません。（習得状況を知りたい。）

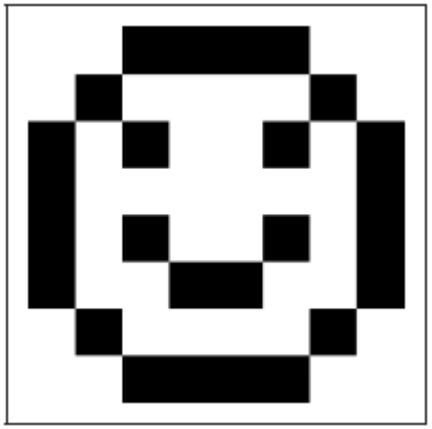
配列による画像の表示

画像の表現



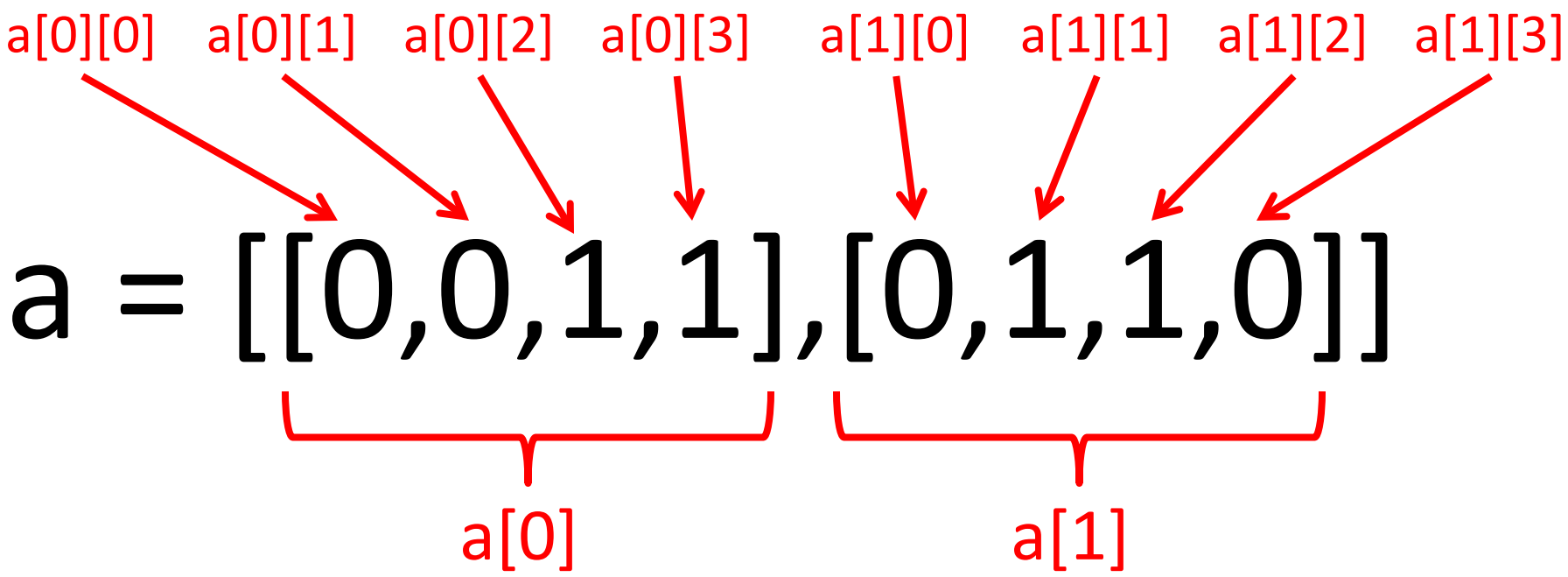
isrb の
プロンプト

```
irb(main):002:0> コントロールD
cm12345$ isrb
>> a = [[0,0,1,1],
?>      [0,1,1,0]]
=> [[0 , 0, 1, 1], [0, 1, 1, 0]]

>> show(a)
=> nil
```



単なる参考

配列の要素



画像の操作

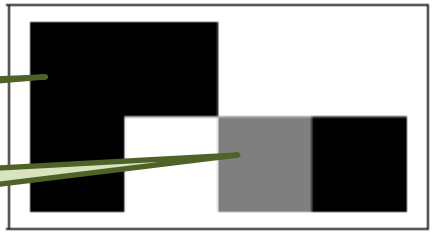
```
>> a[0][0]
=> 0
>> a[0][2]
=> 1

>> a[1][2]=0.5
=> 0.5
>> show(a)
=> nil
```

座標(0,0) の明度を参照

座標(2,0) の明度を参照

座標(2,1) の明度を変更

再表示



練習2.1

- 次のようなデータを作成し、画像として表示させよ。

```
w = [[0 ,1 ,1 ,1 ,1 ,1] ,
     [0 ,1 ,0 ,0 ,0 ,1] ,
     [0 ,1 ,0 ,1 ,0 ,1] ,
     [0 ,1 ,1 ,1 ,0 ,1] ,
     [0 ,0 ,0 ,0 ,0 ,1]]
```

条件分岐と繰り返し

条件分岐 --- 場合分けを使った計算

```
def max(x,y)
  if y < x
    x
  else
    y
  end
end
```

max.rb

```
irb(main):003:0> load("./ max.rb")
=> true
irb(main):004:0> max(123, 456)
=> 456
irb(main):005:0> max(max(12, 34), max(56, 78))
=> 78
```

3 通りの場合分け

```
def sign(x)
    if x < 0
        -1
    else
        if 0 < x
            1          # not(x<0) and 0<x
        else
            0          # not(x<0) and not(0<x)
        end
    end
end
```

sign.rb

複雑な条件

- 色々な比較

書き方	数学	意味
x > y	$>$	x が y より大きい
x >= y	$\geq$	x が y 以上
x == y	$=$	x と y が等しい（x=y でないことに注意）
x < y	$<$	x が y より小さい
x <= y	$\leq$	x が y 以下
x != y	$\neq$	x と y が異なる

- 条件式の組合せ

書き方	意味
x > y || x == 0	x > y または x == 0
x < y && y < z	x < y かつ y < z
!(x < y && y < z)	(x < y かつ y < z) でない

複雑な条件

= でないことに注意
= は代入

- 色々な比較

書き方	数学	意味
x > y	$>$	x が y より大きい
x >= y	$\geq$	x が y 以上
x == y	$=$	x と y が等しい（x=y でないことに注意）
x < y	$<$	x が y より小さい
x <= y	$\leq$	x が y 以下
x != y	$\neq$	x と y が異なる

- 条件式の組合せ

書き方	意味
x > y || x == 0	x > y または x == 0
x < y && y < z	x < y かつ y < z
!(x < y && y < z)	(x < y かつ y < z)でない

どれが正しいか？

x の値が 7、y の値が5、z の値が 3であるとして

1. x < y
2. x <= y
3. y != z
4. z > x
5. z == x

どれが正しいか？

x の値が 7、y の値が5、z の値が 3であるとして

1. x < y
2. x <= y
3. x < y && y != z
4. x <= y || y == z
5. !(x < y && y == z)

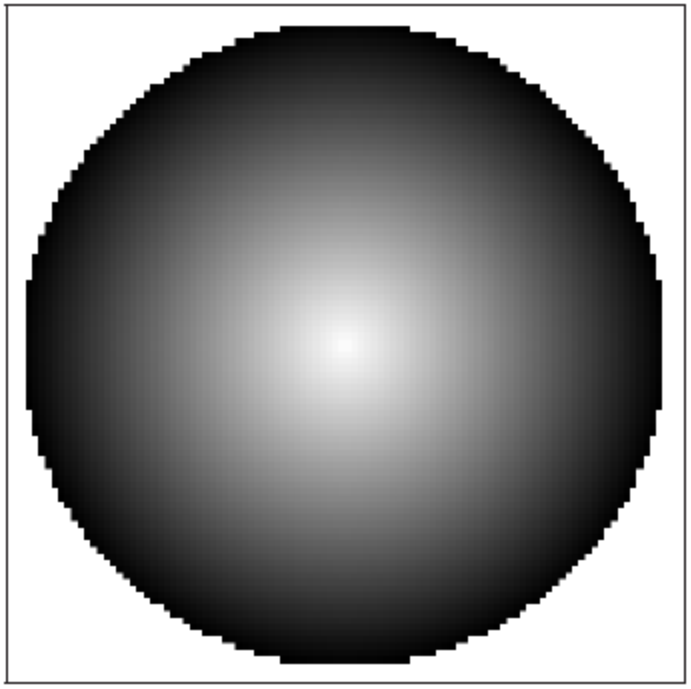
練習3.3c)

c) 3 つの異なる値 x, y, z が与えられたときの中央値を求める median(x,y,z). 中央値とは大きさ順に並べたときに真ん中に来る値のことである。(median.rb というファイルを作れ。)

進捗状況の確認

1. できた（できた時点で投票してください）
2. まだ

繰り返しによる画像の作成



与えられた大きさの 1 次元配列を作る

```
>> image = Array.new(6)
=> [nil , nil , nil , nil , nil , nil]
>> a = Array.new(3)
=> [nil , nil , nil]
```

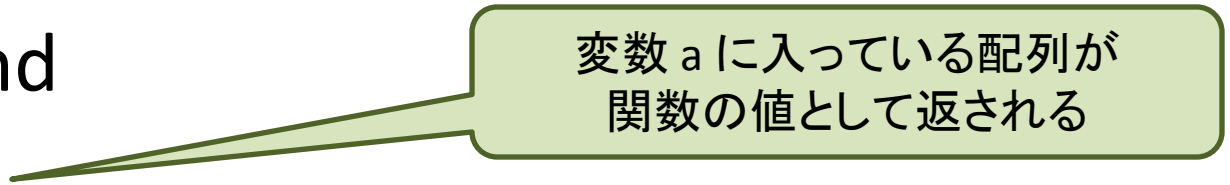
繰り返しによって、
配列の全要素をそれぞれ変更する

```
>> for i in 0..2
>>     a[i] = 0
>> end
=> 0..2
>> a
=> [0, 0, 0]
```

練習（make1d）

- 大きさ n で中身が全て 0 であるような 1 次元配列を作る関数 make1d(n) を定義せよ。（make1d.rb というファイルに格納する。）

```
def make1d(n)
  a = Array.new(?)
  for i in 0..?
    a[?] = ?
  end
  a
end
```



時間がないので…

- 共通資料の配布プログラムから、make1d.rb をダウンロード

```
>> load("./make1d.rb")
=> true
>> make1d(3)
=> [0, 0, 0]
```

2 次元配列を作る

```
>> for i in 0..5
>>     image[i] = make1d(3)
>> end
=> 0..5
>> image
=> [[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0], [0,
0, 0], [0, 0, 0]]
```

6行3列の配列

練習（make2d）

- h 行 w 列の配列を作る make2d(h,w) を定義せよ。ただし、作られる配列の中身は全て 0 にせよ。（もちろん、make1d を使う。make2d.rb というファイルに格納する。）
  - Array.new によって配列を作るのを忘れずに。
  - 最後に配列を関数の値として返すのを忘れずに。

時間がないので…

- 共通資料の配布プログラムから、make2d.rb をダウンロード

```
>> load("./make2d.rb")
=> true
>> make2d(6, 3)
=> [[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0], [0,
0, 0], [0, 0, 0]]
```

2 重の繰り返し

```
def sphere(s)
    image = make2d(s, s)
    for y in 0..(s-1)
        for x in 0..(s-1)
            image[y][x] = b(s, x, y)
        end
    end
    image
end
```

変数 image に入っている配列が
関数の値として返される

sphere.rb

練習3.2

- （sphere.rb をダウンロード。）
- 式(3.2)の計算をする関数 b(s,x,y) を定義せよ。（sphere.rb の中で定義すればよい。）

(r,r) と (x,y) の間の距離

$$b(s,x,y) = \begin{cases} \dfrac{r - d(x,y,r,r)}{r} & (d(x,y,r,r) \le r) \\ 1 & (d(x,y,r,r) > r) \end{cases} \qquad (3.2)$$

（ただし $r = s/2.0$）

- show(sphere(100)) を実行して表示される画像を確めよ。

進捗状況の確認

1. show(sphere(100)) がうまく行った時点で投票してください。
2. show(sphere(100)) がうまく行かない。
3. b が定義できた
4. まだ

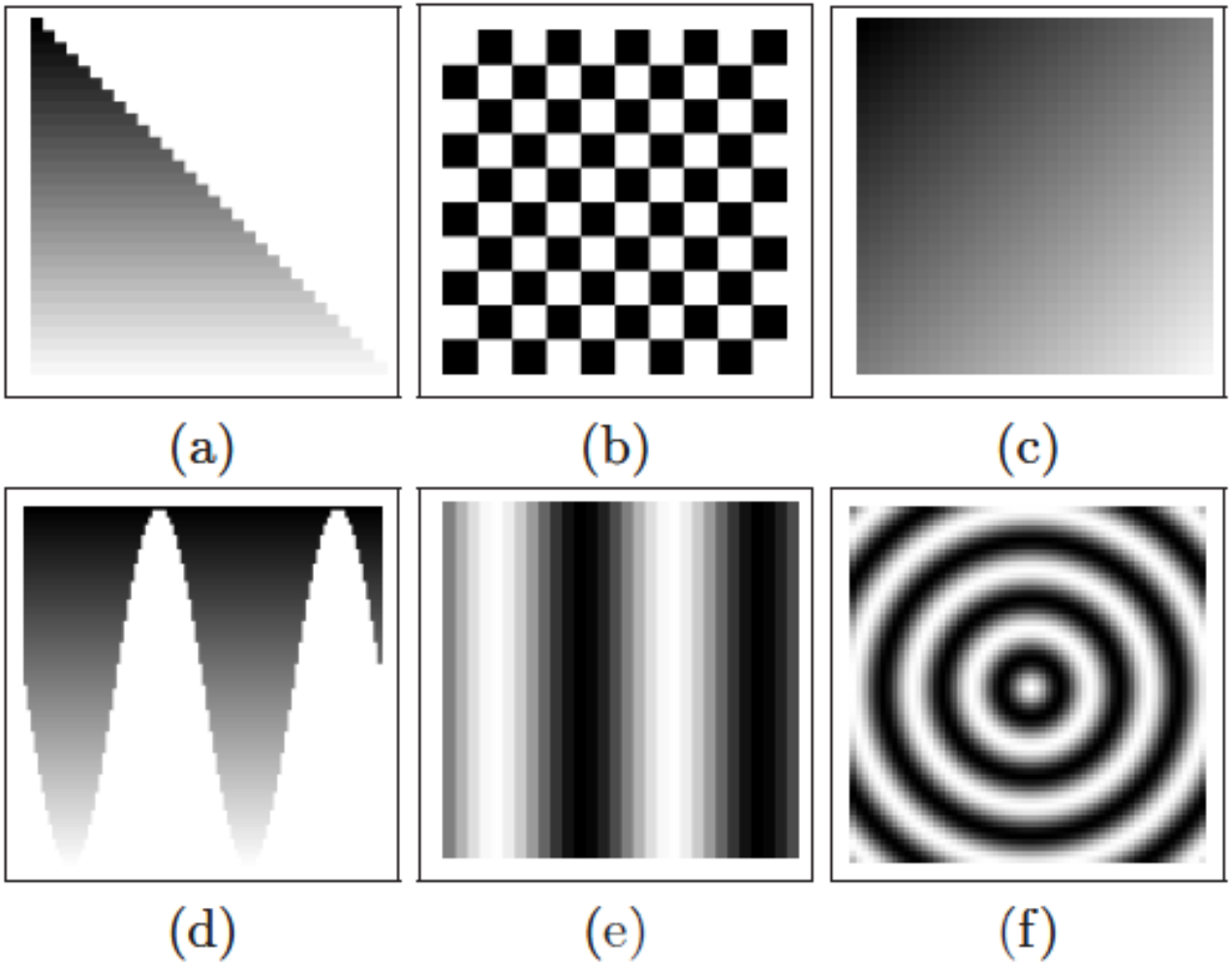
画面のファイルへの保存

- アプリケーションのユーティリティにある「グラブ」を起動する。
- メニューバーの「取り込み」メニューの「スクリーン」を選択し、指示に従って操作する。
- メニューバーの「ファイル」メニューの「保存」を選択する。ファイル名を聞かれるのでファイル名を入力する。特に指定しないと、「書類」のフォルダに保存されるので注意。
- レポートのメールにファイルを添付する。
  - 「Windows 対応の添付ファイルを送信」にチェックが入っていることを確認せよ。

グラブできたか？

1. できた
2. まだ

練習 26: (色々な図形 *) 次のような画像を作成する関数をそれぞれ定義せよ。



レポート

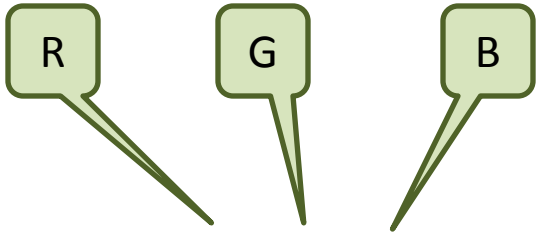
- 課題
  - 自分の好きな図を描いてみよう。プログラムと実行結果の画面を提出してください。
    - 自分で工夫して考えた図を歓迎します。
  - 画面は「グラブ」でファイルに落し、メールに添付してください。（この際、圧縮しないでください。）
- 提出先

  is-komaba@lyon.is.s.u-tokyo.ac.jp
- Subject:欄は、

  is-komaba report 1

  としてください。
- 〆切: 11月15日

カラー画像の表現




```
>> d=[[[0 ,0 ,0] ,[0 ,1 ,0] ,[0 ,0 ,1]] ,
?>      [[1 ,0 ,0] ,[1 ,1 ,0] ,[1 ,0 ,1]]]
=> [[[0 , 0, 0], [0, 1, 0], [0, 0, 1]] , [[1 , 0, 0],
[1, 1, 0], [1, 0, 1]]]
>> show(d)
=> nil
```

3.5　定義のまとめ

条件分岐: 式$_1$ が成り立つときは 式$_2$ を、そうでないときは 式$_3$ を計算する条件分岐は次のように書く。このとき 式$_1$ を**条件式**という。

```
if 式1
     式2
else
     式3
end
```

真偽値: `true` と `false` はそれぞれ、真と偽を表わす値である。

真偽値を与える論理演算

```
irb(main):003:0> x = 3
=> 3
irb(main):004:0> 1 < x
=> true
irb(main):005:0> x == 2
=> false
```

```ruby
def is_even(x)
    x%2 == 0
end
```

is_even.rb

```ruby
load ("./is_even.rb")

def tnpo(n)
    if is_even(n)
        n/2
    else
        3*n + 1
   end
end
```

tnpo.rb

空の値: `nil` は「無い」ことを表わす特別な値である。

配列の作成: `Array.new(`式`)` という式は、大きさが式の値であるような配列を作る。作られた配列の中身はすべて `nil` である。

繰り返し: 変数の値を式$_1$の値から式$_2$の値まで 1 ずつ順に変化させながら、命令$_1$から命令$_n$を毎回実行する**繰り返し**は次のように書く。

```
for 変数 in 式1 .. 式2
  命令1
  ⋮
  命令n
end
```

練習問題確認プログラム

- ex03.rb（練習3.1と3.3-3.9）を実行せよ。（3.7と3.8はオプショナル）
- 今回は提出する必要はありません。