条件分岐

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. z == y
3. 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. 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 && z == x
2. x > y || z == x
3. !(x > y)

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

実数

- 2 次方程式 $ax^2 + bx + c = 0$ の解の個数を求める solutions(a,b,c). 判別式の値だけでなく、1 次方程式になっている場合にも対応せよ。(solutions.rb というファイルを作ろう。)
- 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ろう。)

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
```

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

is_even.rb

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

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

tnpo.rb

2.3　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
```

文字列

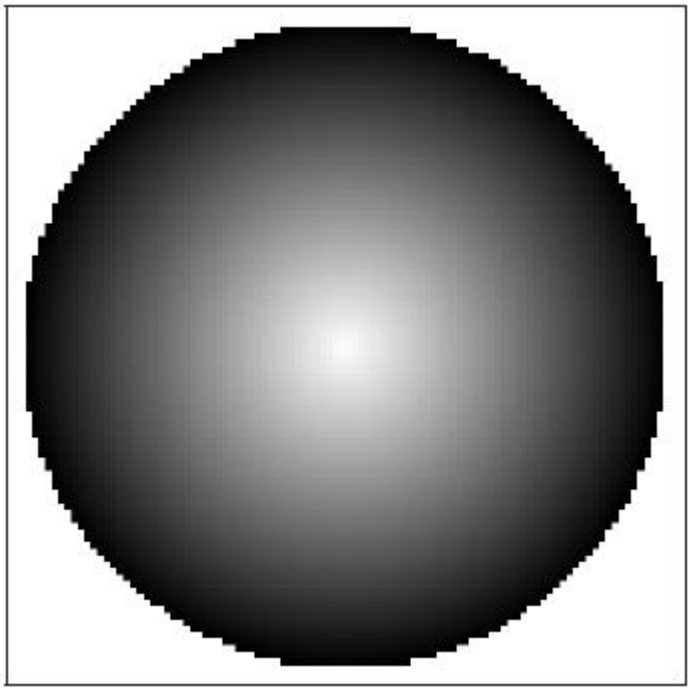
```
irb(main):003:0> s = "abra"
=> "abra"
irb(main):004:0> t = "cadabra"
=> "cadabra"

irb(main):006:0> u = s + t
=> "abracadabra"
irb(main):007:0> "123" + "456"
=> "123456"
```

```
irb(main):009:0> s.length()
=> 4
irb(main):010:0> (s + t).length()
=> 11

irb(main):012:0> s[0..0]
=> "a"
irb(main):013:0> s[1..2]
=> "br"
irb(main):014:0> t[1..(t.length()-1)]
=> "adabra"
```

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

- 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。

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]]
```

練習

- 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。
- 式3.2 の計算をする関数 b(r,x,y) を定義せよ (x, y) だけでなく r も引数となっていることに注意せよ)。

原点と (x,y) との間の距離

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

2 重の繰り返し

```
def sphere(r)
    image = make2d(2*r, 2*r)
    for y in 0..(2*r -1)
        for x in 0..(2*r -1)
            image[y][x] = b(r,x,y)
        end
    end
    image
end
```

sphere.rb

練習

- show(sphere(20)) を実行して表示される画像を確めよ。

次は何を返す？

```
a = Array.new(2)
b = Array.new(2)
for i in 0..1
    b[i] = a
    for j in 0..1
        b[i][j] = i
    end
end
b
```

1. [[0,0],[0,0]]
2. [[0,0],[1,1]]
3. [[0,1],[0,1]]
4. [[1,1],[1,1]]
5. [0,0]
6. [0,1]
7. [1,1]

次は何を返す？

```
b = Array.new(2)
for i in 0..1
    b[i] = Array.new(2)
    for j in 0..1
        b[i][j] = i
    end
end
b
```

1. [[0,0],[0,0]]
2. [[0,0],[1,1]]
3. [[0,1],[0,1]]
4. [[1,1],[1,1]]
5. [0,0]
6. [0,1]
7. [1,1]

3.4　定義のまとめ

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

文字列を作る: 2 つの " 記号の間にある文字や記号は文字列を作る。

文字列の結合: 式$_1$ + 式$_2$ という式は、式$_1$, 式$_2$ の値が文字列のとき、それらの文字列をつなげた文字列を作る。

文字列の長さ: 式$_1$`.length()` という式は、式$_1$ が表わす文字列の長さを求める。

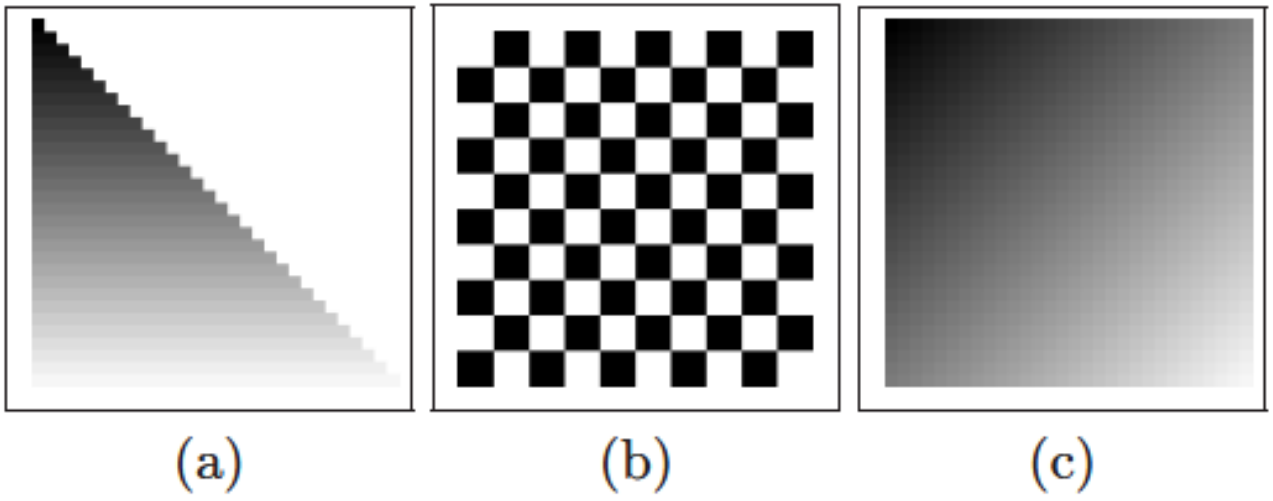
部分文字列: 式$_1$`[`式$_2$`..`式$_3$`]` という式は、式$_1$ が表わす文字列の式$_2$ 番目から式$_3$ 番目までを取り出した文字列を作る。

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全て `nil` である。

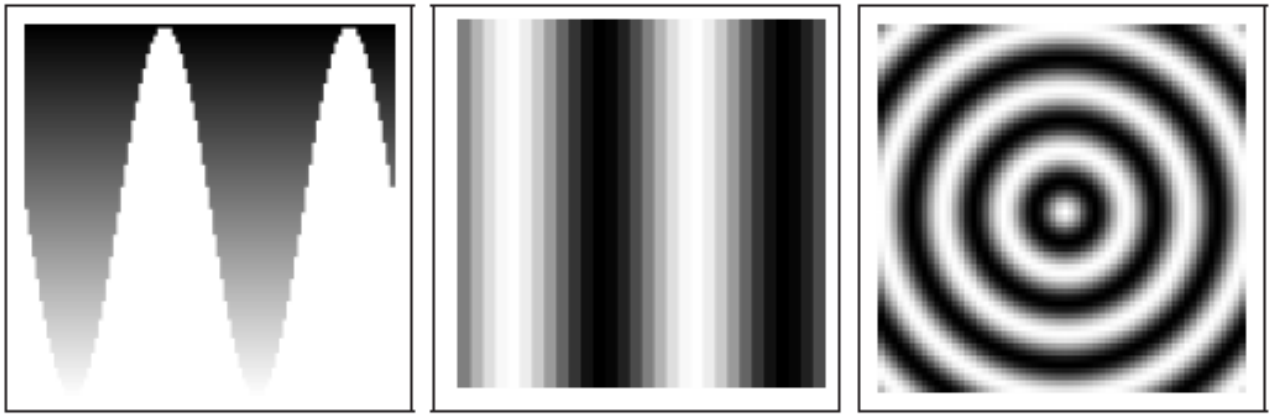
繰り返し: 変数 の値を $式_1$ の値から $式_2$ の値まで 1 ずつ順に変化させながら、$命令_1$ から $命令_n$ を毎回実行する**繰り返し**は次のように書く。

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

練習 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。



(a) (b) (c)



(d) (e) (f)