## 1.4 不等式

不等式[1]は数学を学ぶ上で、と言うよりプログラムを書く上で欠かせないものである。数学でもプログラムでも不等号の使われ方は

$$\text{a) } x > 0、\qquad \text{b) } x < 0、\qquad \text{c) } x \geqq 0、\qquad \text{d) } x \leqq 0$$

のように 4 通りある。

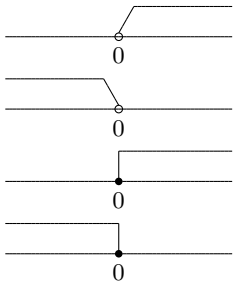
c) と d) の不等号は国際的には $\geq$ と $\leq$ だが、ここでは日本の習慣に従っておこう。郷に入っては郷に従うものだよ。記号の書き方は統一感に欠けるが、記号の意味は厳密である。

a) 「$x$ は 0 より大きい」「$x$ は 0 を超える（0 は含まない）」

b) 「$x$ は 0 より小さい」「$x$ は 0 未満（0 は含まない）」

c) 「$x$ は 0 以上」「$x$ は 0 に等しい、または、$x$ は 0 より大きい」

d) 「$x$ は 0 以下」「$x$ は 0 に等しい、または、$x$ は 0 未満」




不等式は数直線上に “範囲として” 図示されることが多いが、不等式が表すのは範囲ではない。等号がふたつを**同値**関係で結ぶように、不等号はふたつを大小で比較する。“ふたつを比較” するのだ。たとえば 2 と 5 を不等号を用いて比較すれば $2 < 5$ である。$2 \leqq 5$ と書いてもよい。なぜかというと、d) の意味に従えば $2 \leqq 5$ は「$2 = 5$、または、$2 < 5$」と同じことだからだ。“A **または** B” が正しい表現になる場合とは、A、B の少なくとも一方が正しい場合である。$2 = 5$ は正しくないが $2 < 5$ は正しい。よって $2 \leqq 5$ は文句なく正しい表現である。

こんなところが不等式のとっつきにくさを際立たせるのかもしれない。しかし不等式を解くことに限れば、解き方は一点の注意を除いて方程式とまったく同じである。一点の注意とは

**不等式の両辺に負の値を掛ける・割るしたら、不等号の向きを逆にする**

ことである。これは至極当たり前のことで、2 と 5 を比較すると $2 < 5$ であるが、両方の数に $-1$ を掛けた場合は $-2$ と $-5$ の比較になるので $-2 > -5$ である。$2 < 5$ と $-2 > -5$ では、そもそも比較しているものが違う。$3 = 3$ と $-3 = -3$ においても、同値という “関係” が同じなのであって比較対象の値は違っているが、これを上下に

$$\begin{array}{rcr} 3 & = & 3 \\ -3 & = & -3 \end{array}$$

[1]正式には高校 1 年で学習するようだが. . . 。

のように並べると、上の等式の両辺に $-1$ を掛けた結果が下の等式になったように見える。“そう見えること” なら “そうすること” にしてもかまわないだろう。つまり、等式の両辺に $-1$ を掛けてもかまわない。**移項**と呼ばれる操作も、等式の両辺に同じ数を加えた結果、項が符号を変えて移動したように見えるから、そうしてよいのだ。不等式を

$$\begin{array}{rcl} 2 & < & 5 \\ -2 & > & -5 \end{array}$$

のように上下に並べれば、不等式の両辺に $-1$ を掛けたことで不等号の向きが逆になったように見える。だったら、不等号の向きを逆にすれば両辺に $-1$ を掛けてもかまわないのである。

実は $-1$ を掛けることは数の向きを逆にするだけでなく、記号の向きまで逆にするものなのだ。その点、記号はうまくできていて $>$、$<$ は互いに逆向きである。でも、$=$ は逆向きにしても $=$ なのだ。

実際に不等式を解いてみよう。ただひとつの注意を除いて方程式と同じに解けばよい。前回の方程式を不等式に変えて解き直すことにしよう。

a′)
$$\begin{array}{rcll} 6x-7 & > & 11 & \\ 6x & > & 11+7 & \text{（移項：符号を変える）} \\ x & > & \dfrac{18}{6} & \text{（両辺を 6 で割る）} \\ x & > & 3 & \end{array}$$

b′)
$$\begin{array}{rcll} -2x+6 & \geqq & 1 & \\ -2x & \geqq & 1-6 & \text{（移項：符号を変える）} \\ x & \leqq & \dfrac{-5}{-2} & \text{（両辺を } -2 \text{ で割る：不等号の向きを変える）} \\ x & \leqq & \dfrac{5}{2} & \end{array}$$

**ex.** **1** c′) $\dfrac{2}{3}x+8<4$ を解いてみよう。

符号や不等号が変化したりしなかったりだが、「いつ/何を/どうする」かを正確に把握しておけば決して間違えることはない。それはプログラムも一緒だ。プログラムもほんのわずかの間違いで、とんでもない結果を招いてしまう。数学での姿勢が影響するものである。方程式や不等式を解く際は、「移項するとき/項の符号を/変える」「両辺を負の数で乗除したとき/不等号の向きを/逆にする」ことだけが特別のことである。それ以外はいつもの計算と何ら変わらないのだ。だから a′) では 2 行目で移項したときだけ、b′) では移項に加え 3 行目で両辺を $-2$ で割ったときだけ、特別

のことをしている。それ以外は普段通りなのだ。もちろん $c'$) は移項したときだけが特別だって気づいたよね。

さて、不等式の薀蓄（うんちく）はこれぐらいにして、前回方程式を解くプログラムを書いたことを思い出して、不等式を解くプログラムを書いてみよう。

[Inequalities.playground]

```
import UIKit

var a = Int(arc4random()) % 100 - 50; if a == 0 { a = 1 }
var stra = ""
if a == 1 {
    stra = "x"
} else if a == -1 {
    stra = "-x"
} else {
    stra = String(a) + "x"
}
var b = Int(arc4random()) % 100 - 50
var strb = ""
if b > 0 {
    strb = " + " + String(b)
} else if b < 0 {
    strb = " - " + String(-b)
}
var c = Int(arc4random()) % 100 - 50

var comp = "", solx = ""
if arc4random() % 2 == 0 {
    comp = " > "
    if a >= 0 { solx = " の解は、x > " } else { solx = " の解は、x < " }
} else {
    comp = " < "
    if a >= 0 { solx = " の解は、x < " } else { solx = " の解は、x > " }
}
print(stra + strb + comp + String(c) + solx + String(Float(c-b)/Float(a)))
```

前回の [Equations3.playground] に手を加えただけなので一部を書き換えてもよいし、不等式のプログラムとして下の行に追加してもよい。その場合、上の行には方程式のプログラムが記述されたままになっているので、これを放置すると変数を二重に定義することになって具合が悪い。以前のプログラムを消せばスッキリするが、せっかく書いたものを消したくなければ、/* と */ で囲むようにするとよい。たとえば

```
/*
（方程式を解くプログラム）
*/
```

のようにすると、/* から */ までがコメント扱いになる。あとで方程式を解くプログラムを実行したくなったら /* と */ を取り、不等式を解くプログラムのほうを /* と */ で囲めばよい。

不等式を解くプログラムは長いものになったが、不等式にはそれなりの事情があるので止むを得ないだろう。ただし方程式を解くプログラムとの違いは、最後の解の表示部分だけであることに気づくだろう。不等式は、ただひとつの注意を除けば方程式と同じことをするので、その注意が表示形式に反映されただけなのである。

if arc4random() % 2 == 0 は不思議なことをしていると思うだろうが、これは不等式の解法とは無関係の処置である。作られる不等式が常に “>” 向きでは味気ないので、乱数の偶・奇で不等号の向きを変えたに過ぎない。さらに、“≧” などの不等式も作りたいなら、4 で割った余りによって場合分けをするとよいだろう。ただし、そうすると if 文の分岐が 4 通りとなって少々煩（わずら）わしい。そう思ったら if 文の代わりに switch 文を使うとスッキリ書ける。22–28 行の if 文を

```
switch arc4random() % 4 {
case 0: comp = " > "
    if a >= 0 { solx = " の解は、x > " } else { solx = " の解は、x < " }
case 1: comp = " < "
    if a >= 0 { solx = " の解は、x < " } else { solx = " の解は、x > " }
case 2: comp = " >= "
    if a >= 0 { solx = " の解は、x >= " } else { solx = " の解は、x <= " }
case 3: comp = " <= "
    if a >= 0 { solx = " の解は、x <= " } else { solx = " の解は、x >= " }
default: break
}
```

に変えてみよう。分岐が多いときは役に立つ書き方である。≧ と ≦ はキーボードから入力できないので、>= と <= で入力しなくてはいけない。

ここに掲げた switch 文の例は if 文を含んでいて本質が少々ぼやけてしまったかもしれない。switch 文の書式は

```
switch 値 {
   case 値 1:（値 1 のときの処理）
   case 値 2:（値 2 のときの処理）
                  :
   default:（case に合致しないときの処理）
}
```

が基本の形である。分岐を定めるのは “条件式” ではなく “値” であることに注意されたい。ここに書くのは何らかの式なのだが、大雑把に言って等式や不等式ではない計算式なのである。ちな

みに変数 x だけでも計算式と言う。default:は必ず記述しなければならない。場合によってはcase で全ての場合分けが済むかもしれない。そのときは最後の case を default にするか、または default: break とする。

いま示している例は計算式に arc4random() % 4 を使っているので、値の可能性としては 0, 1, 2, 3 の 4 通りである。そこで case 3:を default:にすれば 1 行ケチれるが、値を明示するほうがわかりやすいと思ったのでそうした。好みの問題であろう。

また switch 文には、いたるところに "if a >= 0 { solx = " の解は、..." が見受けられる。同じものがそこ・ここにあるときはうまくまとめられないか考えてみるものよい。たとえばこんな風にするとプログラムはだいぶ短くなる。

[Inequalities2.playground]

```
import UIKit

var a = Int(arc4random()) % 100 - 50; if a == 0 { a = 1 }
var stra = String(a) + "x"
if a == 1 {
    stra = "x"
} else if a == -1 {
    stra = "-x"
}
var b = Int(arc4random()) % 100 - 50
var strb = ""
if b > 0 {
    strb = " + " + String(b)
} else if b < 0 {
    strb = " - " + String(-b)
}
var c = Int(arc4random()) % 100 - 50

var comp = "", solx = "", comp1 = "", comp2 = ""
switch arc4random() % 4 {
    case 0:  comp = " > ";  comp1 = " > ";  comp2 = " < "
    case 1:  comp = " < ";  comp1 = " < ";  comp2 = " > "
    case 2:  comp = " >= "; comp1 = " >= "; comp2 = " <= "
    default: comp = " <= "; comp1 = " <= "; comp2 = " >= "
}
if a >= 0 {solx = " の解は、x" + comp1} else {solx = " の解は、x" + comp2}
print(stra + strb + comp + String(c) + solx + String(Float(c-b)/Float(a)))
```

4 通りの不等式が作れて、なお最初の [Inequalities.playground] より短い。プログラムは短ければよいというものではないが、まったく同じコードが散らばっているのはよくない。なぜなら、修正をする必要が生じたらすべてを正しく直すのは厳しい。

実は、今回の書き換えはちょっとアクロバティックである。次節以降の話になるが、まったく同じコードが複数あるなら関数にまとめてしまうのがよい解決方法だ。関数という言葉は再三登場しているが、あとでじっくり学ぶことになるので、アクロバディックな部分をどう修正するかの指針を示しておこう。

方程式も不等式も、与えられた式の係数・定数を見れば解はすぐわかるのであった。不等式が面倒なのは、$x$ の係数 $a$ の正・負によって解の不等号の向きが変わることである。そこで、あらかじめ不等号の向きの組み合わせを `comp`, `comp1`, `comp2` で決めておき、`a` の正・負で当てはめているのだ。アクロバティックというよりインチキくさい処置である。プログラムの中には裏で胡散臭（うさんくさ）いことをしているものもあるという悪い見本かな。

では、どうするのが正統なのだろう。たとえば

全品 3 割引き　⇔　「$\boxed{\phantom{00}}$円の品は $\boxed{\phantom{00}}$ × 0.7 円」

という関係は、全に500 円のを代入すれば「500 円の品は 350 円」、1200 円のを代入すれば「1200 円の品は 840 円」との返答が得られる。「全品 3 割引き」の表現が、代入する値によらず適切な値を返す仕組みを作っている。この仕組みが関数である。

ならば、`[Inequalities2.playground]` の `print` 文にもその仕組みがあればよいことになる。具体的には

```
fakeprint(係数: 負, 不等号: > )  →  解は、x <  何々
fakeprint(係数: 正, 不等号: >=)  →  解は、x >= 何々
```

みたいなものを想像してもらいたい。関数の考えは数学だけでなくプログラミングにおいても重要である。中 2 数学でしっかり身につけよう。

**ex. 2**　`[Inequalities2.playground]` の解の表示の仕方はもっと工夫できないだろうか。自分なりの方法を試してみよう。