3.1　変数と代入

まず，$\sqrt{3}$を計算します．

```
> sqrt(3)
[1] 1.732051
```

次に，この計算結果を 4 乗することを考えます．計算結果は 1.732051 ですので以下のようにします．

```
> 1.732051^(4)
[1] 9.000004
```

答えが求まりましたが，1.732051 を逐一コピー & ペーストを行った上で処理を行うのは少し手間です．そこで本節では，計算結果を保存する方法として，変数と代入について紹介します．

R では，次に示す形式で数値や計算結果を**変数**という箱に保存することができます．この「変数という箱に保存する」行為を**代入**と呼びます．

変数 <- 数値や計算結果など

たとえば，x という変数に sqrt(3) の計算結果を代入する場合は，次のように記述します．

```
> x <- sqrt(3)
```

計算結果を変数に代入しましたが，このとき，計算結果そのものは画面に表示されません．変数に何が入っているかを確認する場合は，変数名（ここでは x）をそのまま入力します．

```
> x
[1] 1.732051
```

この変数x(中身は$\sqrt{3}$ = 1.732051)を使っていろいろな計算を行うことができます．例えば，x^4 を計算する場合は以下のようにします．

```
> x^(4)
[1] 9
```

3.2　変数名の命名規則

変数名はどのように付けてもよいというわけではなく，次に示す命名規則があります．

- R は大文字と小文字を区別します．
- 変数名にはローマ字や数字を使うことができますが，変数名の先頭を数字にすることはできません．例えば，「1A」という変数名はエラーとなります．
- 以下に挙げる名前は，R 自身の都合で利用が制限されている予約語なので，変数名として用いることはできません．

```
  break  else  FALSE  for   function  if    in     Inf
  NA     NaN   next   NULL  repeat    TRUE  while
```

3.3　変数を使った計算

それでは，変数を使った計算例をいくつか見てみましょう．まず，2 つの変数を用いて演算を行う例です．

```
> x <- 1    # x に 1 を代入する
> y <- 2    # y に 2 を代入する
> x + y     # x の中身と y の中身の和
[1] 3
```

また，1 つの変数に値を代入し直すと，古い値は破棄され，新しい値で上書きされます．

```
> x <- 1    # x に 1 を代入する
> x <- 2    # x に 2 を代入する
> x         # x の値を確認
[1] 2
```

変数に値を代入するだけでは，変数に代入された値が何かは分かりませんでした．しかし，次のように丸括弧 ( ) を用いると，代入と表示を同時に行うことができます．

```
> x <- 1     # 代入のみで結果の表示はない
> ( x <- 1) # 丸括弧で囲むと，代入した結果を表示する
[1] 1
```

練習問題 3.1

〔1〕 変数 x に 2 を代入したあと，$3 + x^2$ を計算してください．

〔2〕 次の文を実行すると，変数 a と変数 b には何が入っているでしょうか．

```
> a <- b <- -1.5
```

〔3〕〔2〕で生成した変数 a について，a の整数部分を求める trunc(a) と，a の切り下げを行う floor(a) を実行してください．trunc(a) と floor(a) の結果の違いは何でしょうか．

〔4〕 スーパーのレジに商品を持って行き，100 円を払ったときのおつりを計算することを考えます．今，変数 value に 23（円）が格納されています．value 円（= 23 円）の商品を購入したときに 100 円玉をレジに渡したときのおつりを変数 remain に代入してください．

〔5〕〔4〕の続きとして，おつり remain（= 77 円）を貰うときの 50 円玉の枚数が何枚であるか，を考えます．前章で紹介した演算子 %/% を用いることで「77 ÷ 50 の商」が「50 円玉の枚数」となり，「77 ÷ 50 の余り」である 27 円が残りの金額となります．よって，「50 円玉の枚数」と「残りの金額を再度，変数 remain に格納」するプログラムは以下となります．

```
M50    <- remain %/% 50
remain <- remain %%  50
```

これを参考に，おつり remain（= 77 円）を貰うときの 10 円玉の枚数，5 円玉の枚数，1 円玉の枚数をそれぞれ変数 M10，M05，M01 に代入するプログラムを作成してください．

3.4 ベクトルについて

ある 5 人の体重（単位 kg）の値，50，55，60，65，70 を変数に代入することを考えます．

```
> a <- 50
> b <- 55
> c <- 60
> d <- 65
> e <- 70
```

ここではデータの数が 5 個なので間に合いましたが，このような変数の使い方では，データの数が 50 個，100 個と増えてしまえば当然対応できなくなります．そもそも R は，データ解析のために作られたソフトウェアで，複数の値をひとまとめに扱う処理を得意としています．R には**ベクトル**というデータ構造があり，それを使うことで複数の数値をまとめて 1 つの変数に代入することができます．

```
> x <- c(50, 55, 60, 65, 70)
```

これで 5 人の体重が 1 つの変数 x に代入されました．c() は複数の値から 1 つのベクトルを作る関数です．試しに x と入力してみると，変数 x の中に確かに 5 人の体重のデータが代入されていることが分かります．

```
> x
[1] 50 55 60 65 70
```

こうして 1 つのベクトルとしてまとめられたデータの合計や平均値を求めることも，R ならば簡単にできます．

```
> ( y <- sum(x) )   # 5 人の体重の合計を変数 y に代入
[1] 300
> y/5               # 5 人の体重の平均値
[1] 60
```

関数 sum() の他にも，ベクトルに対して処理を行う関数がいくつか用意されています．

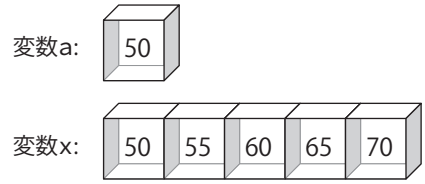
ベクトルに対する関数一覧

関数	意味	関数	意味	関数	意味
cor()	相関係数	median()	中央値	rev()	要素の逆順
cumsum()	累積和	min()	最小値	sd()	標準偏差
diff()	前後の差分	order()	各要素の位置	sort()	要素の整列
length()	要素の数	prod()	要素の掛け算	sum()	合計
max()	最大値	range()	範囲	summary()	要約統計量
mean()	平均値	rank()	各要素の順位	var()	不偏分散

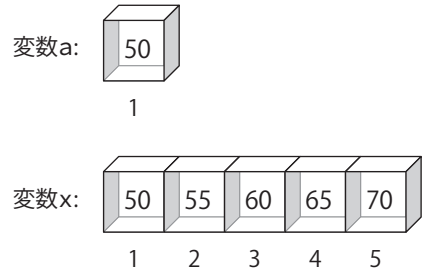
さて，前節で紹介した変数とベクトルがどう違うのか見てみましょう※ 1.

```
> a <- 50                    # 変数
> x <- c(50, 55, 60, 65, 70)  # ベクトル
```

変数 a は箱が 1 つで，値（要素）を 1 つしか代入することができません．一方，ベクトル x は箱が 5 つあり，値（要素）を 5 つまで代入することができます．



箱には左から順番に 1，2，3，……と番号が付いています．



変数は変数名だけで値を表示することができます．一方，ベクトルは変数名を入力すると全ての値が表示され，要素の番号を指定するとその要素の値が表示されます．

※ 1　C 言語や Java などの他の言語をご存知の方は，ベクトルを「配列」のイメージで理解してください．

```
> a      # 変数
[1] 50
> x[2]   # ベクトルの2番目の要素
[1] 55
```

お気付きの方がいるかもしれませんが，ここで以下の命令を実行します．

```
> a[1]
[1] 50
```

ベクトルと同じ方法で，変数 a の値を表示することができました．これまで変数 a とベクトル x は別物のように説明してきましたが，実は，変数 a は「箱が 1 つしかないベクトル」と同じです．

3.5 ベクトルの操作

ベクトルの要素のうち，特定の番号の値を表示する場合は [ ]（かぎカッコ）を使用して以下のようにします．

ベクトル名[番号]

ベクトルのうち，特定の要素の値を変更する場合は以下のようにします．

ベクトル名[変更したい要素の番号] <- 値

ベクトルと 1 つの値や，ベクトルとベクトルを結合するときは，関数 c() を用いて次のようにします．

c(ベクトル，値)
c(ベクトル，ベクトル)

以上について例を示します.

```
> x <- c(50, 55, 60, 65, 70)
> x[2]                # 2 番目の要素を取り出す
[1] 55
> x[3] <- 88          # 3 番目の要素を 88 に変更する
> x                   # x の値を確認する
[1] 50 55 88 65 70
> x <- c(x, 99)       # x に値 99 を結合する
> x                   # x の値を確認する
[1] 50 55 88 65 70 99
```

練習問題 3.2

〔**1**〕以下の 5 つのデータを変数 y に格納してください.

```
1 2 3 4 5
```

〔**2**〕変数 y の値と変数 Y（大文字）の値を表示してください.

〔**3**〕〔2〕で作成した変数 y の合計を，関数を使って求めてください.

〔**4**〕変数 y の後ろに，値 9 を結合し，結果を変数 z に格納してください.

〔**5**〕y の 6 番目の値と，〔4〕で作成した変数 z の 6 番目の値を表示してください.

3.6　おまけ

〔**1**〕1，2，……，10 と，1 から順に 1 ずつ増えるようなベクトルを生成する場合は，

```
> x <- c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

としてもよいですが，:（コロン）を用いることで次のように簡単に生成することができます.

```
> x <- 1:10
```

逆に，1 ずつ減るようなベクトルを生成する場合は次のようにします．

```
> x <- 10:1
```

〔2〕1，1，……，1 と，1 が 10 個並んでいるようなベクトルを生成する場合は，

```
> x <- c(1, 1, 1, 1, 1, 1, 1, 1, 1, 1)
```

としてもよいですが，関数 rep(**値,繰り返し数**) を用いることで次のように簡単に生成することができます．

```
> x <- rep(1, 10)
```

〔3〕ここで，規則性のあるベクトルを生成する命令をまとめておきます．

規則性のあるベクトルを生成する命令一覧

命令	意味
1:10	1 から 10 まで 1 ずつ増えるベクトル
10:1	10 から 1 まで 1 ずつ減るベクトル
seq(1, 10, by=2)	1 から 10 まで 2 ずつ増加する等差数列を生成する
rep(1, 10)	値 1 を 10 回繰り返した数列を生成する
rep(x, 10)	ベクトル x を 10 回繰り返した数列を生成する

〔4〕まず，以下を実行してベクトル x を生成します．

```
> ( x <- c(1, 2, 3, 4, 5) )
[1] 1 2 3 4 5
```

例えば，x[2] とするとベクトル x の 2 番目の要素を表示しますが，[ ] の中にベクトルを指定することもできます．以下を実行すると，ベクトル x の 2 番目と 4 番目の要素を表示します．

```
> x[c(2,4)]
[1] 2 4
```

また，[ ] の中に負の値を指定することもできます．例えば，-2 を指定すると，ベクトル x の 2 番目以外の要素を表示します．

```
> x[-2]
[1] 1 3 4 5
```

[ ] の中に負の値のベクトルを指定することもできます．以下を実行すると，ベクトル x の 2 番目と 4 番目以外の要素を表示します．

```
> x[c(-2,-4)]
[1] 1 3 5
```

〔**5**〕関数 names() で，ベクトルにラベルをつけることができます．

```
> ( x <- c(1, 2, 3) )
[1] 1 2 3
> names(x) <- c("1番目", "2番目", "3番目")
> x

1番目 2番目 3番目
    1     2     3
```

練習問題 3.3

〔**1**〕101, 102, ……, 200 と，101 から順に 1 ずつ増えるようなベクトル y を生成してください．

〔**2**〕変数 y のうち奇数番目の要素のみを取り出し，変数 z に代入してください．

〔**3**〕変数 z の長さ（要素の数）を求めてください．