

関数と制御構造

第3講 - 関数の定義とプログラムの作成

村田 昇

講義概要

- R 言語における関数
- 引数の扱い方 (引数名・順序・既定値)
- 自作関数の定義
- 制御構造 (条件分岐・繰り返し)

R 言語における関数

関数 (復習)

- 関数の取り扱いは一般的な計算機言語とほぼ同様
- 関数は引数とその値を指定して実行
- 引数名は順序を守れば省略可能
- 関数の呼び出し方の例

```
f(arg1 = value1, arg2 = value2) # 擬似コード
#' arg1, arg2 は引数の名前, value1, value2 は引数に渡す値を表す
f(value1, value2) # 上と同値, 順序に注意
```

- 引数がない場合もある

関数の実行例

- 正弦関数の計算

```
sin(x = pi/2) # "引数名 = 値" で指定
sin(pi/2) # 上と同値 (引数と値の関係が明かなら引数名は省略可能)
```

- 対数関数の計算

```
help(log) # ヘルプを表示して使い方を確認する
x <- 16; b <- 2 # xやbに適当な数値を代入する. 1行で複数の処理を行う場合は ; を用いて並べる
log(x = x, base = b) # 底をbとする対数
log(x, b) # 上と同値
log(base = b, x = x) # 上と同値
log(b, x) # 上と異なる (=log(x = b, base = x))
log(x) # 自然対数 (既定値による計算 =log(x, base = exp(1)))
```

引数と返値

- ヘルプにより関数の引数および返値を確認できる
 - 引数については“Arguments”の項を参照
 - 返値については“Values”の項を参照
- 引数を省略すると既定値 (default) が用いられる
- ヘルプによる関数仕様の表示の例

```
#' 正規乱数を生成する関数
help(rnorm) # Help タブから指定しても良い
#' ヒストグラムを表示する関数
?hist # help(hist) と同値
```

既定値を持つ関数の実行例 (1/2)

- 正規乱数の生成

```
rnorm(5) # 平均 0 分散 1 の正規乱数を 5 個生成
rnorm(5, mean = 10) # 平均 10 分散 1 の正規乱数を 5 個生成
rnorm(sd = 0.1, n = 5) # 平均 0 分散 0.01 の正規乱数を 5 個生成
rnorm(n = 5, mean = 2, sd = 2) # 平均 2 分散 4 の正規乱数を 5 個生成
```

```
[1] -0.38046043  1.02207181  0.45856968  1.15044014 -0.03996123
[1] 11.903802   9.593342  11.025405   9.274189  10.664963
[1] -0.06512622  0.11532188 -0.06692962 -0.10033647  0.04259349
[1]  2.71065387  4.42052141 -0.69420948  1.91140561 -0.08730777
```

既定値を持つ関数の実行例 (2/2)

- ヒストグラムの表示

```
foo <- rnorm(n = 10000, mean = 50, sd = 10) # 平均 50 標準偏差 10 の正規乱数
hist(foo) # データ以外全て既定値で表示
hist(foo, # 既定値のいくつかを変更する
      breaks = 30, # ビンを 30 程度に調整する
      col = "lightgreen", # 色の指定
      main = "mathematics", # タイトルの指定
      xlab = "score") # x 軸ラベルの指定
#' Plots タブで描画結果を確認
```

実習

練習問題

- ヘルプ機能 (Help タブの検索窓, 関数 help(), ?) を用いて関数 sample() を調べてみよう
- サイコロを 1 回振る試行を模擬してみよう
- サイコロを 10 回振る試行を模擬してみよう
 - 引数 replace を調べよ
- 1 が出やすいサイコロを作ってみよう
 - 引数 prob を調べよ
- 1 から 6 をランダムに並べ替えてみよう

関数の定義

自作関数

- 他の言語と同様に R でも関数を定義できる
- 関数の定義には関数 `function()` を利用する

```
#' 関数 function() 記法 (擬似コード)
関数名 <- function(引数){ # 計算ブロックの開始
  #' このブロック内に必要な手続きを記述する. 複数行に渡って構わない
  return(返値) # 計算結果を明示的に示す
} # ブロックの終了
```

自作関数の例 (1/2)

- 半径 r から球の体積と表面積を求める関数

```
foo <- function(r){
  volume <- (4/3) * pi * r^3 # 球の体積
  surface <- 4 * pi * r^2    # 球の表面積
  out <- c(volume, surface) # 返り値のベクトルを作る
  names(out) <- c("volume", "surface_area") # 返り値の要素に名前を付ける
  return(out) # 値を返す
}
foo(r = 2) # 実行
foo(3)
```

```
volume surface_area
33.51032      50.26548
volume surface_area
113.0973      113.0973
```

自作関数の例 (2/2)

- 初項 a 公比 r の等比数列の最初の n 項 (既定値は 5)

```
bar <- function(a, r, n = 5){
  out <- a*r^(1:n-1) # 1:n-1 と 1:(n-1) は異なるので注意
  return(out) # 値を返す
}
bar(1, 2) # 初項 1 公比 2 の最初の 5 項
bar(1, 2, 10) # 初項 1 公比 2 の最初の 10 項
bar(n = 10, 1, 2) # 変数名を指定すると引数の位置を変えることができる
bar(r = 0.5, n = 10, a = 512) # 同上
```

```
[1] 1 2 4 8 16
[1] 1 2 4 8 16 32 64 128 256 512
[1] 1 2 4 8 16 32 64 128 256 512
[1] 512 256 128 64 32 16 8 4 2 1
```

実習

例題

- 三角形の 3 辺の長さ x, y, z を与えると面積 S を計算する関数を作成せよ.
 - 参考: **ヘロンの公式** より

$$S = \sqrt{s(s-x)(s-y)(s-z)}, \quad s = \frac{x+y+z}{2}$$

が成り立つ.

- 解答例

```
my_heron <- function(x, y, z){
  #' 関数名は上書きされるので独特の名前にするのがお薦め
  s <- (x+y+z)/2 # 補助変数 s の計算
  S <- sqrt(s*(s-x)*(s-y)*(s-z)) # ヘロンの公式による面積の計算
  return(S) # 面積を返す
}
my_heron(3, 4, 5) # よく知られた直角三角形を使って計算結果を確認する
my_heron(12, 13, 5)
```

```
[1] 6
[1] 30
```

練習問題

- 1 から整数 n までの和を求める関数を作成せよ
 - 関数 `sum()` を調べよ (`help(sum)`)
 - 等差数列の和を利用してもよい
- 整数 n の階乗 $n!$ を求める関数を作成せよ
 - 関数 `prod()` を調べよ (`help(prod)`)

制御構造

制御文

- 最適化や数値計算などを行うためには、条件分岐や繰り返しを行うための仕組みが必要となる
- R 言語を含む多くの計算機言語では
 - `if` (条件分岐)
 - `for` (繰り返し・回数指定)
 - `while` (繰り返し・条件指定)
 などの **制御文** が利用可能

if 文

- 条件 A が **真** のとき処理 X を実行する

```
if(条件 A) {処理 X} # 括弧内は複数行に渡ってよい
```

- 上記の if 文に条件 A が **偽** のとき処理 Y を実行することを追加する

```
if(条件 A) {処理 X} else {処理 Y} # 複数行で記述する場合は改行位置に注意
```

if 文の例

- 今日の日付が 19 で割り切れるか?

```

today <- 20240426 # 今日の日付
if(today %% 19 == 0) { # %% は余りを計算
  print("割り切れます。商は以下の値です。")
  print(today %/% 19) # 商を表示
} else { # {}で囲まれたブロックが1つのまとまった処理に対応する
  print("割り切れません。余りは以下の値です。")
  print(today %% 19) # 余りを表示
}

```

```

[1] "割り切れません。余りは以下の値です。"
[1] 11

```

for 文

- ベクトル V の要素を **順に** 変数 i に代入して処理 X を繰り返し実行する

```

for(i in V) {処理 X} # V の要素はどんなデータ型でも良い

```

- 処理 X は変数 i によって実行内容が変わってよい

for 文の例

- アルファベットの 20,15,11,25,15 番目を表示

```

print(LETTERS) # LETTERS ベクトルの内容を表示
for(i in c(20,15,11,25,15)) {
  print(LETTERS[i]) # 順番に表示
}

```

```

[1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q" "R" "S" "T"
[21] "U" "V" "W" "X" "Y" "Z"
[1] "T"
[1] "O"
[1] "K"
[1] "Y"
[1] "O"

```

while 文

- 条件 A が **真** である限り処理 X を繰り返す

```

while(条件 A) {処理 X}

```

- 処理 X は繰り返し必要な実行内容を記述し、終了するときに条件 A が満たされなくなるように書く
- 参考: repeat 文というものもあるので調べてみよう

while 文の例

- 素因数分解する

```

(n <- 2*11*17*31) # 分解の対象。今日の日付や my_fact(5) などでも試してみよ
p <- 2 # 最初に調べる数
while(n != 1){ # 商 (for 文の中で計算している) が 1 になるまで計算する
  if(n%p == 0) { # 余りが 0 か確認
    print(p) # 割り切った数を表示
    n <- n/p # 商を計算して分解の対象を更新
  } else {
    p <- p+1 # 割り切れない場合は次の数を調べる
  } # 更新される p は素数とは限らないのに上手く動く理由を考えてみよう
}

```

```
}
```

```
[1] 11594  
[1] 2  
[1] 11  
[1] 17  
[1] 31
```

実習

例題

- 制御構造を利用して非負の整数 n の階乗 $n!$ を求める関数を作成せよ。ただし、関数 `prod()` を用いないこと。
- `for` 文を用いた解答例

```
my_fact1 <- function(n){  
  val <- 1 # 初期値の代入  
  for(i in 1:n){ # 1 から n まで順に掛ける  
    val <- val*i  
  }  
  return(val) # 計算結果を返す  
}  
my_fact1(4) # 正しい  
my_fact1(3) # 正しい  
my_fact1(2) # 正しい  
my_fact1(1) # 正しい  
my_fact1(0) # 間違い (0!=1)
```

```
[1] 24  
[1] 6  
[1] 2  
[1] 1  
[1] 0
```

- `if` 文を用いた修正版

```
my_fact2 <- function(n){  
  if(n==0){ # n=0 か確認して分岐する  
    return(1)  
  } else {  
    val <- 1  
    for(i in 1:n){ val <- val*i }  
    return(val)  
  }  
}  
my_fact2(4) # 正しい  
my_fact2(3) # 正しい  
my_fact2(2) # 正しい  
my_fact2(1) # 正しい  
my_fact2(0) # 正しい
```

```
[1] 24  
[1] 6  
[1] 2  
[1] 1  
[1] 1
```

- `while` 文を用いた解答例

```

my_fact3 <- function(n){
  val <- 1 # 初期値の代入
  while(n>0){ # nから1まで順に掛ける. nが0なら計算しない
    val <- val*n
    n <- n-1
  }
  return(val)
}
my_fact3(4) # 正しい
my_fact3(3) # 正しい
my_fact3(2) # 正しい
my_fact3(1) # 正しい
my_fact3(0) # 正しい

```

```

[1] 24
[1] 6
[1] 2
[1] 1
[1] 1

```

練習問題

- 整数 n の Fibonacci 数を求める関数を作成せよ
 - Fibonacci 数は以下の漸化式で計算される

$$\begin{aligned}
 F_0 &= 0 \\
 F_1 &= 1 \\
 F_n &= F_{n-1} + F_{n-2}
 \end{aligned}$$

- 行列 X が与えられたとき、各列の平均を計算する関数を作成せよ
- 前問で X がベクトルの場合にはその平均を計算するように修正せよ
 - 関数 `is.vector()` が利用できる

次回の予定

- データフレームの操作
- ファイルの取り扱い
- データの集計