

R 中数据结构

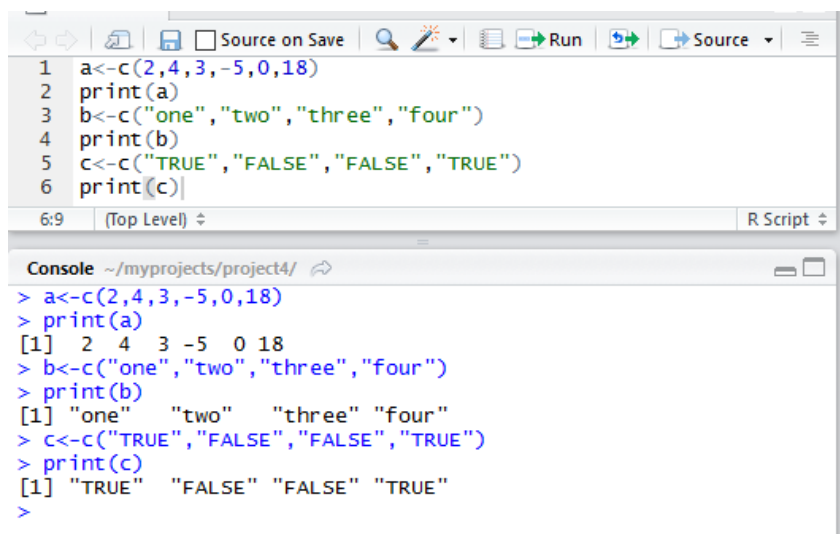
1. 向量

1.1.向量的概念：向量是用于存储数值型、字符型、或逻辑型数据的一维数组。

1.2.向量可存储的数据类型：数值型、字符型、逻辑型。

1.3.特点：单个向量中的数据必须拥有相同的类型或模式。

1.4.R 中向量的创建：执行组合功能的函数 `c()`。如下图：



```
1 a<-c(2,4,3,-5,0,18)
2 print(a)
3 b<-c("one","two","three","four")
4 print(b)
5 c<-c("TRUE","FALSE","FALSE","TRUE")
6 print(c)
```

6:9 (Top Level) R Script

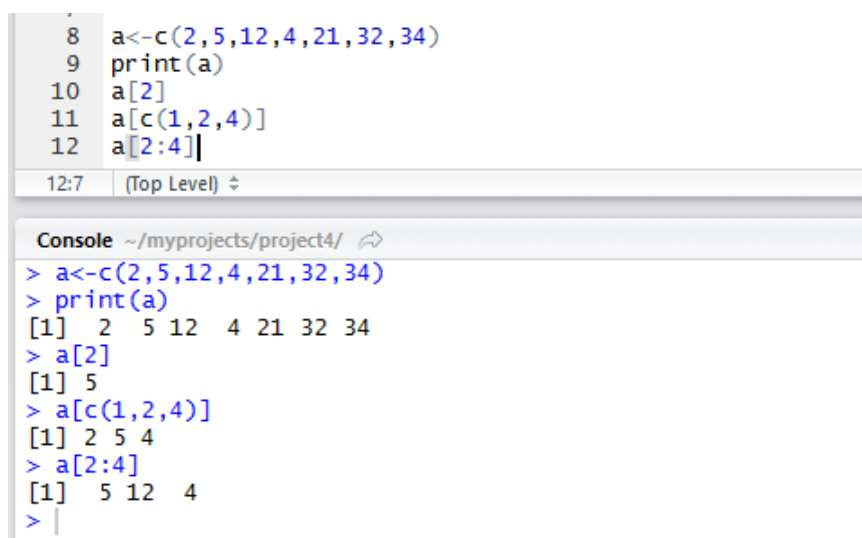
Console ~/myprojects/project4/

```
> a<-c(2,4,3,-5,0,18)
> print(a)
[1] 2 4 3 -5 0 18
> b<-c("one","two","three","four")
> print(b)
[1] "one" "two" "three" "four"
> c<-c("TRUE","FALSE","FALSE","TRUE")
> print(c)
[1] "TRUE" "FALSE" "FALSE" "TRUE"
>
```

1.5.访问向量中元素的方法：

- (1) `a[2]` 访问向量 `a` 中的第 2 个元素。
- (2) `a[c(1,2,4)]` 访问向量 `a` 中的第 1, 2, 4 个个元素。
- (3) `a[2:4]` 访问向量 `a` 中第 2 个到第 4 个元素。

如下图：



```
8 a<-c(2,5,12,4,21,32,34)
9 print(a)
10 a[2]
11 a[c(1,2,4)]
12 a[2:4]
```

12:7 (Top Level)

Console ~/myprojects/project4/

```
> a<-c(2,5,12,4,21,32,34)
> print(a)
[1] 2 5 12 4 21 32 34
> a[2]
[1] 5
> a[c(1,2,4)]
[1] 2 5 4
> a[2:4]
[1] 5 12 4
>
```

2.矩阵

2.1.矩阵的概念：矩阵是由数值型、字符型、逻辑型数据组成的二维数组。

2.2.矩阵可存储的数据类型：数值型、字符型、逻辑型。

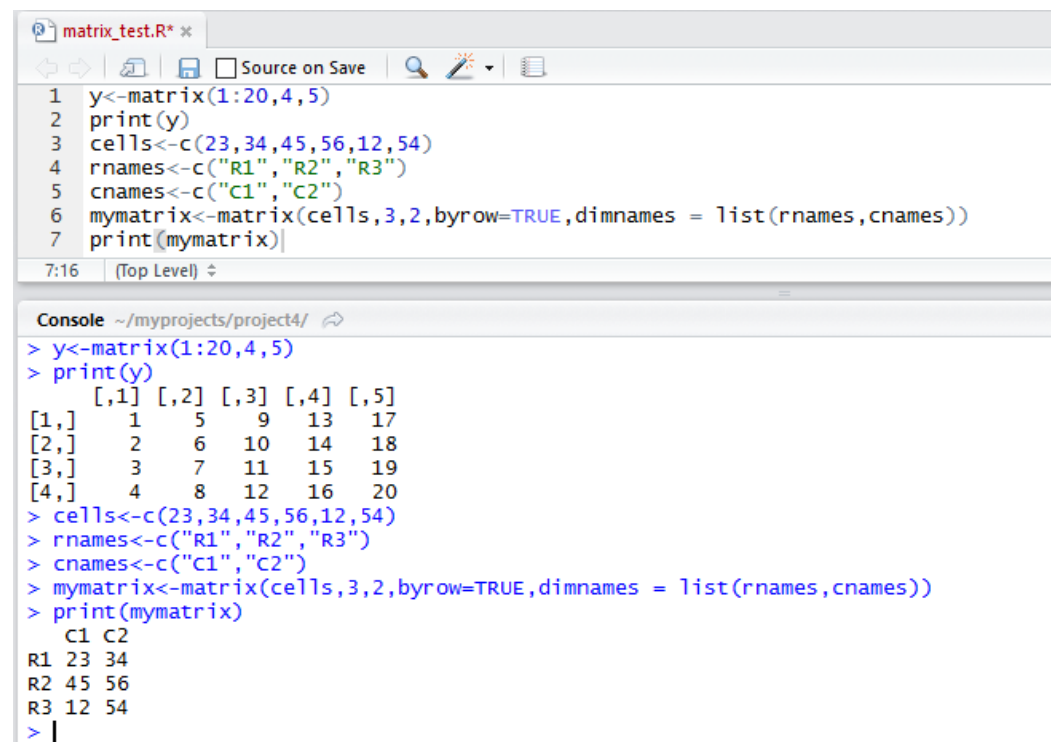
2.3.特点：单个矩阵中的数据必须拥有相同的模式。

2.4.R 中矩阵的创建：使用 `matrix()` 函数。

`mymatrix<-matrix(vector,nrow,ncol,byrow,dimnames=list())`

其中 `vector` 包含矩阵的元素，`nrow` 和 `ncol` 指定矩阵行和列的维数，`byrow` 是个逻辑型数据，为 `TRUE` 时按行填充，为 `FALSE` 时按列填充，不写时默认按列填充。

`Dimnames` 包含了可选的已字符型向量表示的行名和列名。



```
1 y<-matrix(1:20,4,5)
2 print(y)
3 cells<-c(23,34,45,56,12,54)
4 rnames<-c("R1","R2","R3")
5 cnames<-c("C1","C2")
6 mymatrix<-matrix(cells,3,2,byrow=TRUE,dimnames = list(rnames,cnames))
7 print(mymatrix)
```

7:16 (Top Level) ↕

Console ~/myprojects/project4/ ↗

```
> y<-matrix(1:20,4,5)
> print(y)
      [,1] [,2] [,3] [,4] [,5]
[1,]    1     5     9    13    17
[2,]    2     6    10    14    18
[3,]    3     7    11    15    19
[4,]    4     8    12    16    20
> cells<-c(23,34,45,56,12,54)
> rnames<-c("R1","R2","R3")
> cnames<-c("C1","C2")
> mymatrix<-matrix(cells,3,2,byrow=TRUE,dimnames = list(rnames,cnames))
> print(mymatrix)
      C1 C2
R1 23 34
R2 45 56
R3 12 54
> |
```

2.5.读取矩阵中的元素

- (1) 取一个元素：`x[2,3]`（取第 2 行第 3 列元素）
- (2) 取多个元素：`x[1,c(2,3)]`或 `x[c(1,2, 3),c(1,4)]`（逗号前代表行数，后代表列）
- (3) 取一行元素：`x[2,]`
- (4) 取一列元素：`x[,1]`

如下图：

```
9 x<-matrix(1:20,5,4)
10 x
11 x[1,2]
12 x[1,c(2,3)]
13 x[c(2,3,5),c(2,3)]
14 x[1,]
15 x[,2]
16

14:6 (Top Level) ↕

Console ~/myprojects/project4/ ↗
> x<-matrix(1:20,5,4)
> x
      [,1] [,2] [,3] [,4]
[1,]    1    6   11   16
[2,]    2    7   12   17
[3,]    3    8   13   18
[4,]    4    9   14   19
[5,]    5   10   15   20
> x[1,2]
[1] 6
> x[1,c(2,3)]
[1] 6 11
> x[c(2,3,5),c(2,3)]
      [,1] [,2]
[1,]    7   12
[2,]    8   13
[3,]   10   15
> x[1,]
[1] 1 6 11 16
> x[,2]
[1] 6 7 8 9 10
> |
```

3.数组

3.1.数组概念：数组类似矩阵，用于存储数值型、字符型、逻辑型数据，维度大于2。

3.2.数组可存储的数据类型：数值型、字符型、逻辑型。

3.3.特点：数组同向量和矩阵一样，单个数组中的数据只能拥有一种模式。

3.4.R 中数组创建：

`myarray<-array(vector,dimensions,dimnames)`

`vector` 包含了数组中的元素，`dimensions` 是一个数值型向量，给出各个维度下标的最大值。`dimnames` 是各维度名称标签列表，选填。

```
1 dim1<-c("A1","A2")
2 dim2<-c("B1","B2","B3")
3 dim3<-c("C1","C2","C3","C4")
4 z<-array(1:24,c(2,3,4),dimnames=list(dim1,dim2,dim3))
5 z
6:1 (Top Level) ⚙
```

```
Console ~/myprojects/project4/ ↗
> dim1<-c("A1","A2")
> dim2<-c("B1","B2","B3")
> dim3<-c("C1","C2","C3","C4")
> z<-array(1:24,c(2,3,4),dimnames=list(dim1,dim2,dim3))
> z
, , C1
      B1 B2 B3
A1    1  3  5
A2    2  4  6

, , C2
      B1 B2 B3
A1    7  9 11
A2    8 10 12

, , C3
      B1 B2 B3
A1   13 15 17
A2   14 16 18

, , C4
      B1 B2 B3
A1   19 21 23
A2   20 22 24

> |
```

3.5.取数组中元素：与矩阵中读取元素的方法类似，只是多一个维度。

4.数据框

4.1.数据框概念：数据框与数据库中数据集类似，是 R 中最常处理的数据结构。

4.2.可存储的数据类型：数值型、字符型、逻辑型。

4.3.特点：数据框中可同时存在不同的数据类型，每一列的数据类型一致。

4.4.R 中数据框的创建：

`mydata<-data.frame(coll,col2,col3...)` coll..代表一个列向量，可为任意类型。

如下图：

```
dataframe_test.R*
1 patientID <- c(1, 2, 3, 4)
2 age <- c(25, 34, 28, 52)
3 diabetes <- c("Type1", "Type2", "Type1", "Type1")
4 status <- c("Poor", "Improved", "Excellent", "Poor")
5 patientdata <- data.frame(patientID, age, diabetes, status)
6 patientdata

2:20 (Top Level)

Console ~/myprojects/project4/
> patientID <- c(1, 2, 3, 4)
> age <- c(25, 34, 28, 52)
> diabetes <- c("Type1", "Type2", "Type1", "Type1")
> status <- c("Poor", "Improved", "Excellent", "Poor")
> patientdata <- data.frame(patientID, age, diabetes, status)
> patientdata
  patientID age diabetes    status
1         1  25    Type1     Poor
2         2  34    Type2  Improved
3         3  28    Type1  Excellent
4         4  52    Type1     Poor
> |
```

4.5.读取数据框中元素:

- (1) mydata[1], mydata[1:2]
- (2) mydata[c(1,3,4)]#取值大于等于2时需要用向量形式表示,不可 mydata[1,3]。
- (3) mydata ["x1"], mydata[c("x1","x3")]
- (4) mydata\$x1
- (5) attach()和 detach()

attach(mydata)

print(x1)

detach(mydata)

使用这两个函数时遇到一个问题:

```
Source on Save | Run | 
1 patientID <- c(1, 2, 3, 4)
2 age <- c(25, 34, 28, 52)
3 diabetes <- c("Type1", "Type2", "Type1", "Type1")
4 status <- c("Poor", "Improved", "Excellent", "Poor")
5 patientdata <- data.frame(patientID, age, diabetes, status)
6 patientdata
7 attach(patientdata)
8 status
9 detach(patientdata)

9:20 (Top Level) ▾

Console ~/myR/ ↻
> attach(patientdata)
The following objects are masked _by_ .GlobalEnv:
  age, diabetes, patientID, status

The following objects are masked from patientdata (pos = 3):
  age, diabetes, patientID, status

> status
[1] "Poor"      "Improved"   "Excellent"  "Poor"
> detach(patientdata)
> |
```

原因在于 `attach()` 和 `detach()` 函数可以将数据框添加到搜索路径当中，从而简便变量的提取。但是，如果环境中存在相同的变量名称，数据框中的变量会被原始变量覆盖。而本程序之前已经定义了全局变量 `age`, `diabetes`, `patientID`, `status`。

以下便不会有问题：

```
16 a<-data.frame(x1=c(1,2,3),x2=c(87,89,98))
17 attach(a)
18 x1
19 detach(a)|

19:10 (Top Level) ▾

Console ~/myR/ ↻
> a<-data.frame(x1=c(1,2,3),x2=c(87,89,98))
> attach(a)
> x1
[1] 1 2 3
> detach(a)
> |
```

(6) with()

`with(mydata, print(x1))` 可换成任意函数。

若有多条语句需要加 `{}`，且各语句之间需要换行，不能用逗号或空格隔开。

```
with(mydata, { print(x1)
               mean(x1) })
```

注：R 语言中是区分大小写的，像 `c`, `x`, `s` 等这种大小写区别不大的字母要尤其注意，否则程序会报不明确的错误。

```
Source on Save
1 x<-data.frame(x1=c(1,2,3,4),x2=c("L1","L2","L3","L4"),x3=c(89,97,92,87))
2 with(x,{print(x3)
3   mean(x3)}})

3:13 (Top Level)

Console ~/myprojects/project4/
> x<-data.frame(x1=c(1,2,3,4),x2=c("L1","L2","L3","L4"),x3=c(89,97,92,87))
> with(x,{print(x3)
+   mean(x3)}})
[1] 89 97 92 87
[1] 91.25
> |
```

5.因子

5.1.因子概念：R 中名义型（类别）变量和有序型变量（有序类别）都称为因子。在 R 语言中，因子（factor）表示的是一个符号、一个编号或者一个等级，即，一个点。

5.2.可存储的数据类型：字符型

5.3.特点：名义型变量是没有顺序之分的类别变量，有序型变量则是有等级之分的。如，一班，二班，三班是名义型变量；优良中差则是有序型变量。它们都称为因子。

5.4.R 中因子的创建：

首先定义一个向量，然后使用函数 `factor()` 将向量转化为因子。

`factor(vector,ordered=TRUE/FALSE,levels=c())`

其中，`ordered` 取值为 `TRUE` 时，为有序型变量，`levels` 可指定类别顺序，`levels` 未进行定义时，系统会自动根据字母进行排序。`ordered` 默认取值为 `FALSE`，此时为名义型变量。

```
1 diabetes <- c("Type1", "Type2", "Type1", "Type1")
2 status <- c("Poor", "Improved", "Excellent", "Poor")
3 diabetes <- factor(diabetes)
4 diabetes
5 status <- factor(status, order = TRUE)
6 status
7
7:1 (Top Level) ⚡

Console ~/myR/ ↗
> diabetes <- c("Type1", "Type2", "Type1", "Type1")
> status <- c("Poor", "Improved", "Excellent", "Poor")
> diabetes <- factor(diabetes)
> diabetes
[1] Type1 Type2 Type1 Type1
Levels: Type1 Type2
> status <- factor(status, order = TRUE)
> status
[1] Poor Improved Excellent Poor
Levels: Excellent < Improved < Poor
> |
```

可以看出因子的值除了返回字符型向量外，还包含类型的所有取值，有序型变量会标示出等级顺序。等级顺序未指定时按照首字母排序，Levels: Excellent < Improved < Poor。添加等级顺序时则严格按照指定的顺序。下图展示了两者的区别

```
1 diabetes <- c("Type1", "Type2", "Type1", "Type1")
2 status <- c("Poor", "Improved", "Excellent", "Poor")
3 status1 <- c("Poor", "Improved", "Excellent", "Poor")
4 status <- factor(status, ordered = TRUE)
5 status
6 status2 <- factor(status1, ordered = TRUE, levels = c("Poor", "Improved", "Excellent"))
7 status2
8
8:1 (Top Level) ⚡ R Script ⚡

Console ~/myR/ ↗
> diabetes <- c("Type1", "Type2", "Type1", "Type1")
> status <- c("Poor", "Improved", "Excellent", "Poor")
> status1 <- c("Poor", "Improved", "Excellent", "Poor")
> status <- factor(status, ordered = TRUE)
> status
[1] Poor Improved Excellent Poor
Levels: Excellent < Improved < Poor
> status2 <- factor(status1, ordered = TRUE, levels = c("Poor", "Improved", "Excellent"))
> status2
[1] Poor Improved Excellent Poor
Levels: Poor < Improved < Excellent
>
```

6.列表

6.1.列表概念：列表就是一些对象的有序集合。

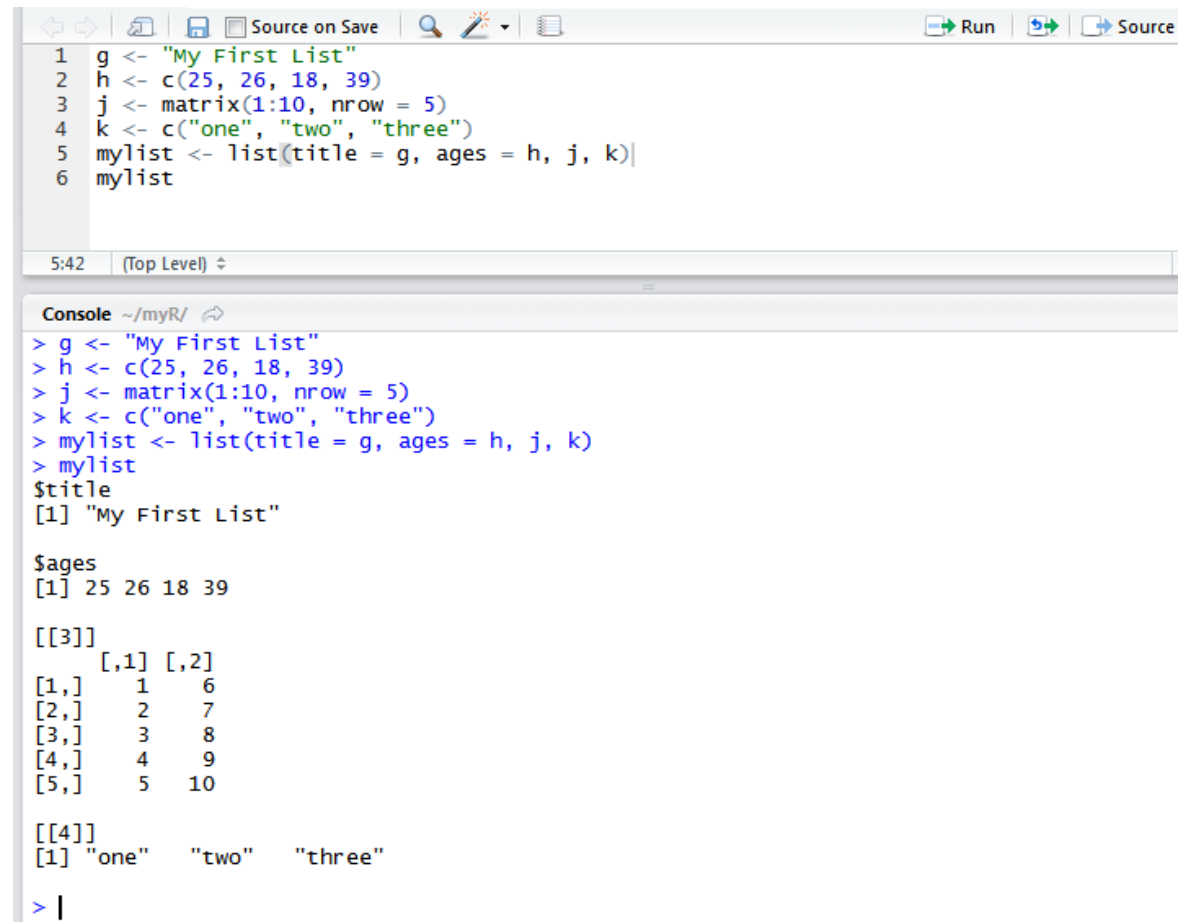
6.2.列表存储的数据类型：列表中可以有向量，矩阵，数据框，数组，列表等。

6.3.特点：列表可以理解为特殊的向量，只是向量中的元素模式必须单一，而列

表中的各元素可以是多种类型。

6.4.R 中列表的创建：

`Mylist<-list(obj1,obj2,obj3,...)` #obj1 代表列表中的元素，它可以是任意类型。



```
1 g <- "My First List"
2 h <- c(25, 26, 18, 39)
3 j <- matrix(1:10, nrow = 5)
4 k <- c("one", "two", "three")
5 mylist <- list(title = g, ages = h, j, k)
6 mylist
```

5:42 (Top Level) ↕

Console ~/myR/ ↻

```
> g <- "My First List"
> h <- c(25, 26, 18, 39)
> j <- matrix(1:10, nrow = 5)
> k <- c("one", "two", "three")
> mylist <- list(title = g, ages = h, j, k)
> mylist
$title
[1] "My First List"

$ages
[1] 25 26 18 39

[[3]]
  [,1] [,2]
[1,]   1   6
[2,]   2   7
[3,]   3   8
[4,]   4   9
[5,]   5  10

[[4]]
[1] "one" "two" "three"

> |
```

6.5.读取列表中的元素：与数据框类似：

- (1) `mylist[1]` #取列表中第一个元素
- (2) `mylist[1:2]` #取列表中第 1 到第 2 个元素
- (3) `mylist[c("ages","title")]` #取列表中名字为 `ages` 和 `title` 的元素。

注：在下图程序中，列表 `mylist` 中的元素只有前两个元素指定了名称，故读取元素时可以根据元素名称读取，而变量 `j` 和 `k` 不可用 `mylist["j"]` 这样的方式读取。

```
1 g <- "My First List"
2 h <- c(25, 26, 18, 39)
3 j <- matrix(1:10, nrow = 5)
4 k <- c("one", "two", "three")
5 mylist <- list(title = g, ages = h, j, k)
6 mylist[1]
7 mylist[c(2,4)]
8 mylist[1:2]
9 mylist[c("ages", "title")]
```

9:26 (Top Level) ↕

Console ~/myprojects/project4/ ↗

```
> mylist <- list(title = g, ages = h, j, k)
> mylist[1]
$title
[1] "My First List"

> mylist[c(2,4)]
$ages
[1] 25 26 18 39

[[2]]
[1] "one" "two" "three"

> mylist[1:2]
$title
[1] "My First List"

$ages
[1] 25 26 18 39

> mylist[c("ages", "title")]
$ages
[1] 25 26 18 39

$title
[1] "My First List"
```