**单词**

prompt /prɒmpt/ 提示

concatenating /kɒnˈkætɪˌneɪt/连接

**R语言学习笔记**

“ > ” R prompt ，代码输入提示符，属于在右边的代码叫做expression

" <-" assignment operator

“[1]” index, 索引标记符，[1] 表示紧随其后的值（10）是向量中的第一个元素。如果这个向量非常长，超过了一行可以显示的长度，那么你会在新行看到更多的索引标记，比如 [6]，表示该行的第一个值是向量中的第六个元素。

c( ) 函数是一种非常基本和常用的函数，用于创建向量或将多个值合并成一个向量。这里的 c 代表 "combine"（合并）或 "concatenate"（连接）。

y <- c(1.7, "a") *# character*

vector( ) function

x <- vector("numeric", length = 10) # “向量的类型”，向量的长度，输出默认数字是0

class(x)显示对象的类型

x <- 0:6

> class(x)

coercion /kəʊˈɜːʃ(ə)n/强制转换

你在一个向量中混合使用不同类型的对象时，R会进行类型转换（coercion），以确保向量中的所有元素都是同一类型。

implicit coercion隐式强制转换：R在没有明确指示的情况下自动进行的类型转换。R会尽量以合理的方式来表示向量中的所有对象。

"Explicit coercion"（显式强制转换）

是指在编程中，程序员明确地指定一个值或对象从一种数据类型转换为另一种数据类型的过程。与隐式强制转换（implicit coercion）不同，显式强制转换需要程序员直接使用代码进行类型转换的操作。

在R语言中，显式强制转换常见的例子包括使用函数 as.numeric( ), as.character( ), as.integer( ) 等来转换数据类型。例如，如果你有一个字符向量，你需要将它转换为数值型，你可以使用 as.numeric( ) 函数来实现转换：

> x <- 0:6

> class(x)

[1] "integer"

> as.numeric(x)

[1] 0 1 2 3 4 5 6

> as.logical(x)

[1] **FALSE TRUE TRUE TRUE TRUE TRUE TRUE**

> as.character(x)

1. "0" "1" "2" "3" "4" "5" "6"

x <- 1:6

> class(x)

[1] "integer"

> as.numeric(x)

[1] 1 2 3 4 5 6

> class(x)

[1] "integer"

> x <- as.numeric(x)(记得更新x, 否则最好还是integer）

> class(x)

[1] "numeric"

> x

[1] 1 2 3 4 5 6

**factor （）：“因子”函数，是一种用于处理分类数据的特殊类型的向量**

> x <- factor(c("yes", "yes", "no", "yes", "no"))

> x

[1] yes yes no yes no #显示的是因子内部的值

Levels: no yes #level 显示的是因子内部可能出现的值

> table(x) #显示的是因子内部数值出现的频率

x

no yes

2 3

> *## See the underlying representation of factor*

> unclass(x) ##因子（Factor）\*\*是一种用于处理分类数据的特殊类型的向量。虽然因子可以表示像文本这样的分类数据（比如 "yes" 和 "no"），但在内部，它们实际上是通过整数来存储的。

[1] 2 2 1 2 1

attr(,"levels")

1. "no" "yes"

NA (Not Available):

NA 用于表示缺失值或不存在的值。

它可以用于任何类型的向量，包括数值型、字符型、逻辑型等。

NA 通常是由于数据收集过程中的缺失，或者在数据处理过程中未定义的值而产生。

在逻辑测试中，NA 会产生 NA（比如 NA == NA 是 NA，而不是 TRUE 或 FALSE）。

NaN (Not a Number):

NaN 表示不是一个数的数值结果，是一个特定于数值的概念。

NaN 通常是由不确定的数学运算产生的，例如 0/0。

NaN 在数值计算中是重要的，因为它表示了一个无法定义的数值。

在逻辑测试中，NaN 被视为 FALSE（例如 NaN == NaN 是 FALSE）。

总结来说，NA 用于表示缺失的或不可用的数据，可以应用于任何类型的数据，而 NaN 是一种特殊的数值，用于表示无法确定或定义的数值计算结果。在数据分析和处理中，正确理解和处理这两种值是非常重要的。

data.frame( ) 函数

在R语言中用于创建数据框（data frame）。数据框是R中最常用的数据结构之一，类似于电子表格或SQL表，它以列的形式存储数据，每列可以包含不同类型的数据（数值、字符、逻辑值等），但同一列中的所有数据必须是同一类型。

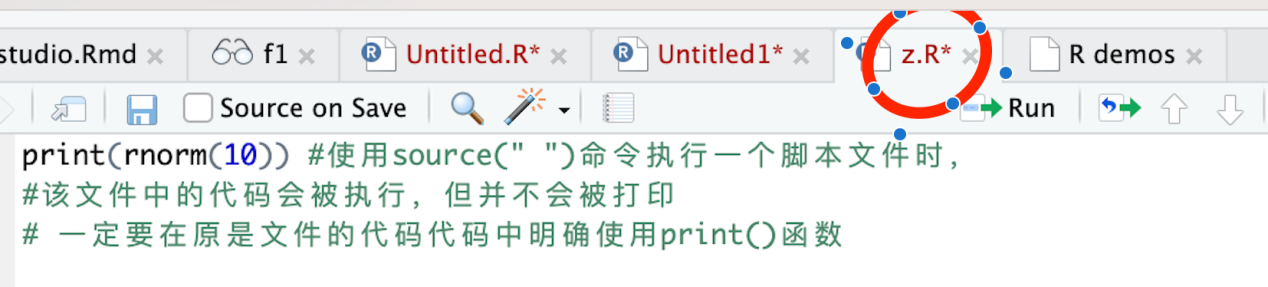
data.frame(column1 = data1, column2 = data2, ...)

column1, column2, ... 是数据框的列名。

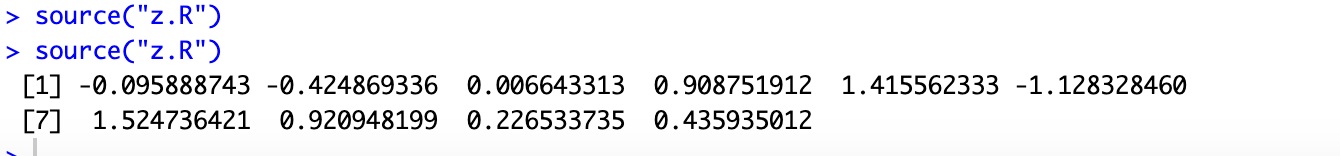
**目录的变更/创建/删除**



**source（） 函数的使用**



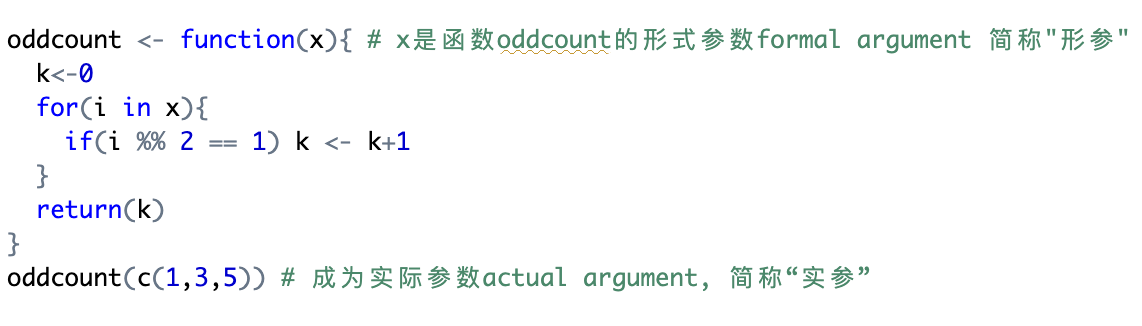
运行结果



data( ) 函数

**data()**函数用于加载R包中内置（自带）的数据集，用于学习数据分析、统计方法或者测试算方法。

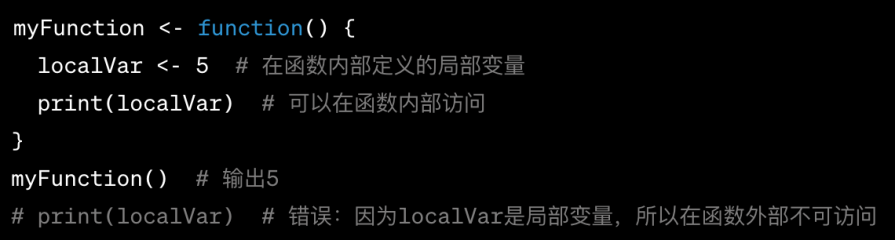
编程语言术语



编程语言术语

局部变量（Local Variables）：

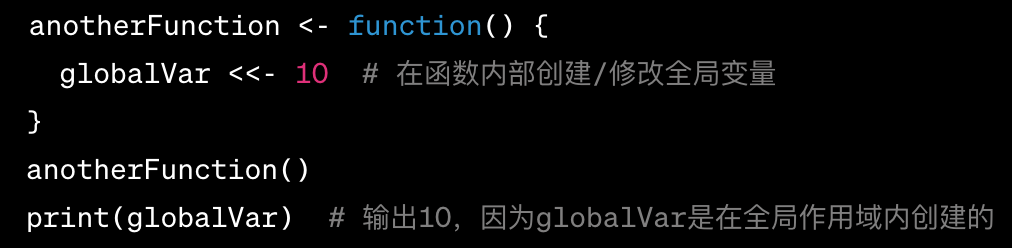
在函数内部定义并且只能在该函数内部访问的变量。这意味着局部变量的作用域被限制在了它被定义的函数内，函数外部不能访问这个局部变量。局部变量的生命周期仅限于函数执行的时间段；当函数执行完毕，局部变量就会被销毁。



全局变量（Global Variables）：

在函数外部定义的变量，可以在程序的任何地方被访问和修改（包括函数内部）。全局变量的作用域是全局的，它们在整个R脚本或命令行会话中都是可用的。全局变量在它们被创建的时刻开始存在，并持续到R会话结束或被显式地删除。

也可以在函数内部使用“<< ” 创建全局变量，但是不建议，容易导致全局变量被污染

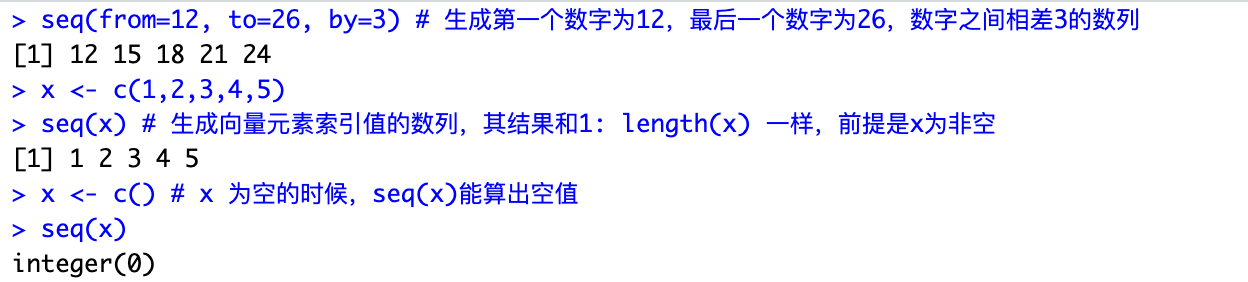


因此最佳实践是尽可能地使用局部变量，或者当确实需要修改全局变量时，明确地指示这一点。使用全局变量可以方便数据在不同函数间的共享，但过度使用全局变量可能会导致代码难以理解和维护，尤其是在大型项目中。

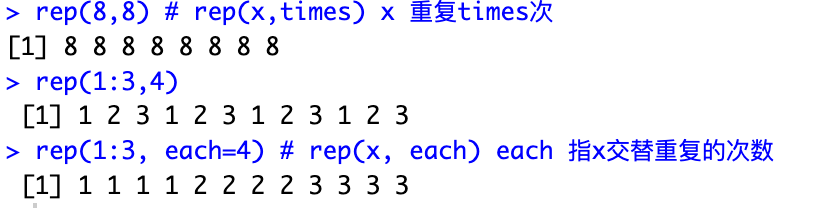
help 函数寻找帮助

1. help (seq) 参数为函数
2. ？seq. ? 后面直接输入要查询的函数
3. 注意：特殊符号以及特殊字符的查询，需要用引号： ?”for”, ?”<”

seq( )生成等差数列函数



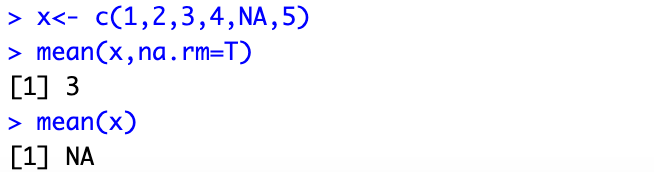
rep( )复制函数



在编写代码，尤其是在数据分析和处理领域时，需要注意以下几个关键点以提高代码效率和性能

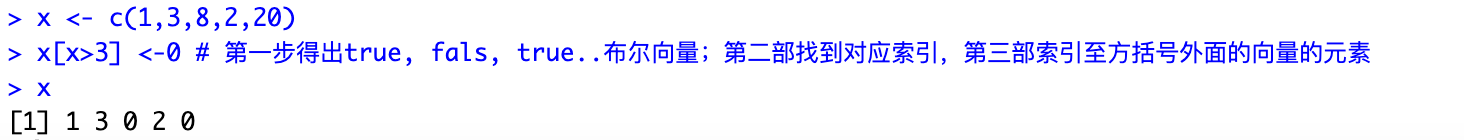
1. **避免在循环中不断增长向量**：尽量避免在循环中使用c()函数或类似的操作来不断增加向量的长度，因为这样会导致频繁的内存分配和数据复制，从而降低代码效率。
2. **预先分配内存空间**：如果可能，最好预先分配足够大的内存空间给向量或其他数据结构。这样，在填充数据时就不需要不断地重新分配和复制数据。例如，如果你知道向量将包含1000个元素，可以先创建一个相应长度的向量，然后填充它。
3. **使用更高效的数据结构**：在某些情况下，选择更合适的数据结构可以提高性能。例如，如果你需要频繁地修改数据集的大小，使用列表（list）可能比向量（vector）更高效。
4. **利用向量化操作**：R语言特别擅长处理向量化操作。尽可能利用这一特性，通过向量化函数代替循环，可以显著提高代码的执行速度。
5. 

na.rm=T, 跳过缺失值NA

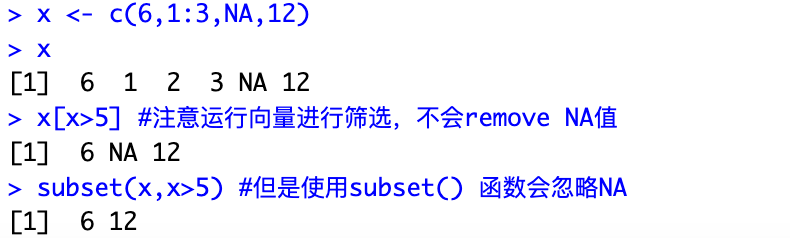


R语言的筛选

1. **运用向量的索引进行筛选**



1. **运用subset( ) 函数进行筛选**



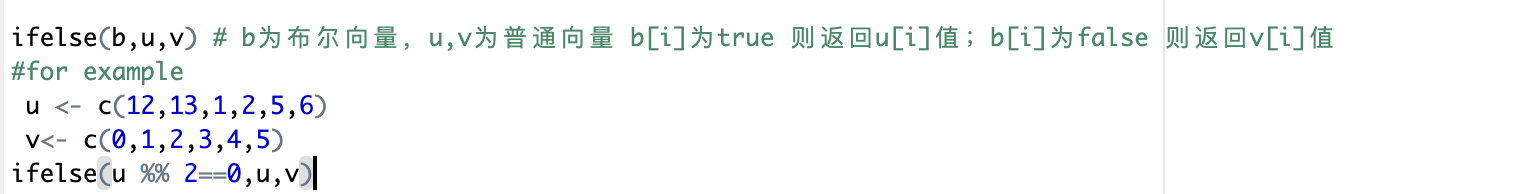
1. **运用which（ ）函数进行筛选得出符合条件的索引值**



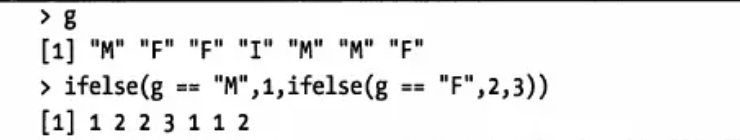
[ 1 ]的奇怪用法



向量化的ifelse（ ）函数



用ifelse ( ) 函数对数据进行编码



在R语言中，v[-1] 代表去掉第一个向量剩余元素

使用单方括号[ ]和双方括号[ [ ] ]提取列表元素的区别

- 使用单方括号[ ]

当你使用单方括号[ ]提取列表中的元素时，结果仍然是一个列表。即使你提取单个元素，提取出来的结果也会被保留为一个长度为1的列表。这种方式允许你同时提取多个元素，返回一个包含这些元素的新列表。

x[1:3]

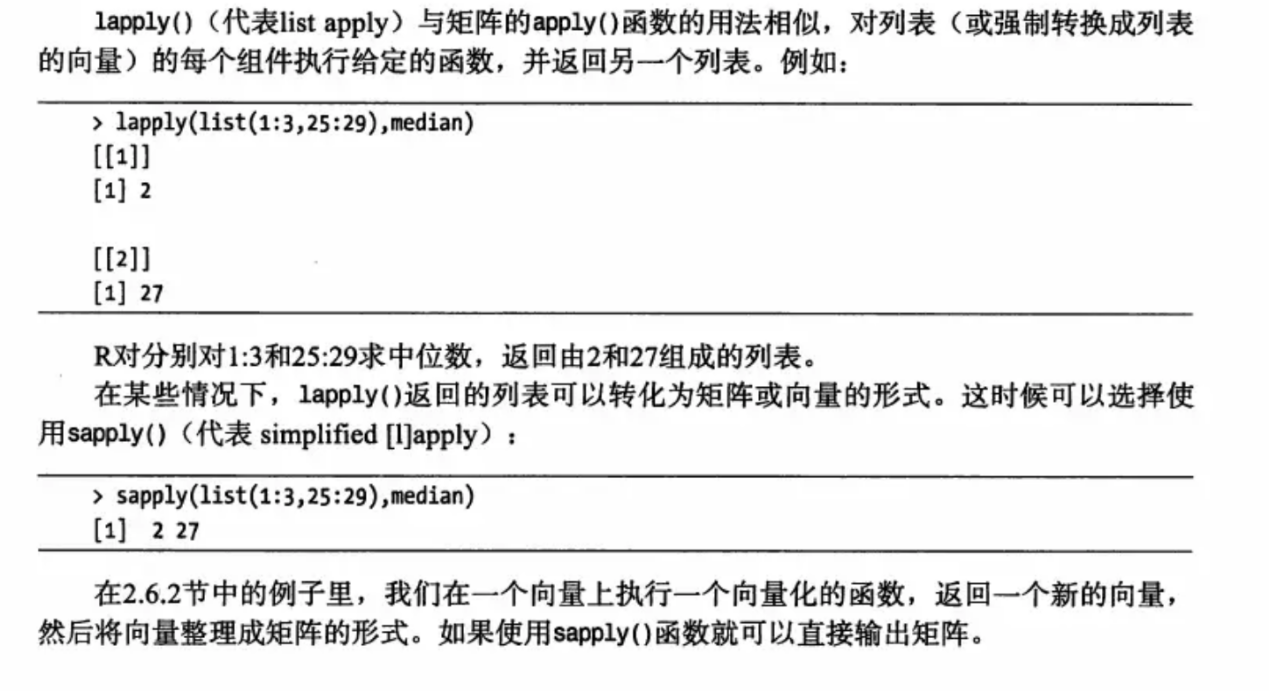
- 使用双方括号[[ ]]

使用双方括号[[ ]]提取列表中的元素时，结果是列表中该位置的元素本身，而不再是列表。这意味着如果列表中的元素是向量或函数等，使用双方括号提取会直接返回该向量或函数，而不是一个包含该向量或函数的列表。

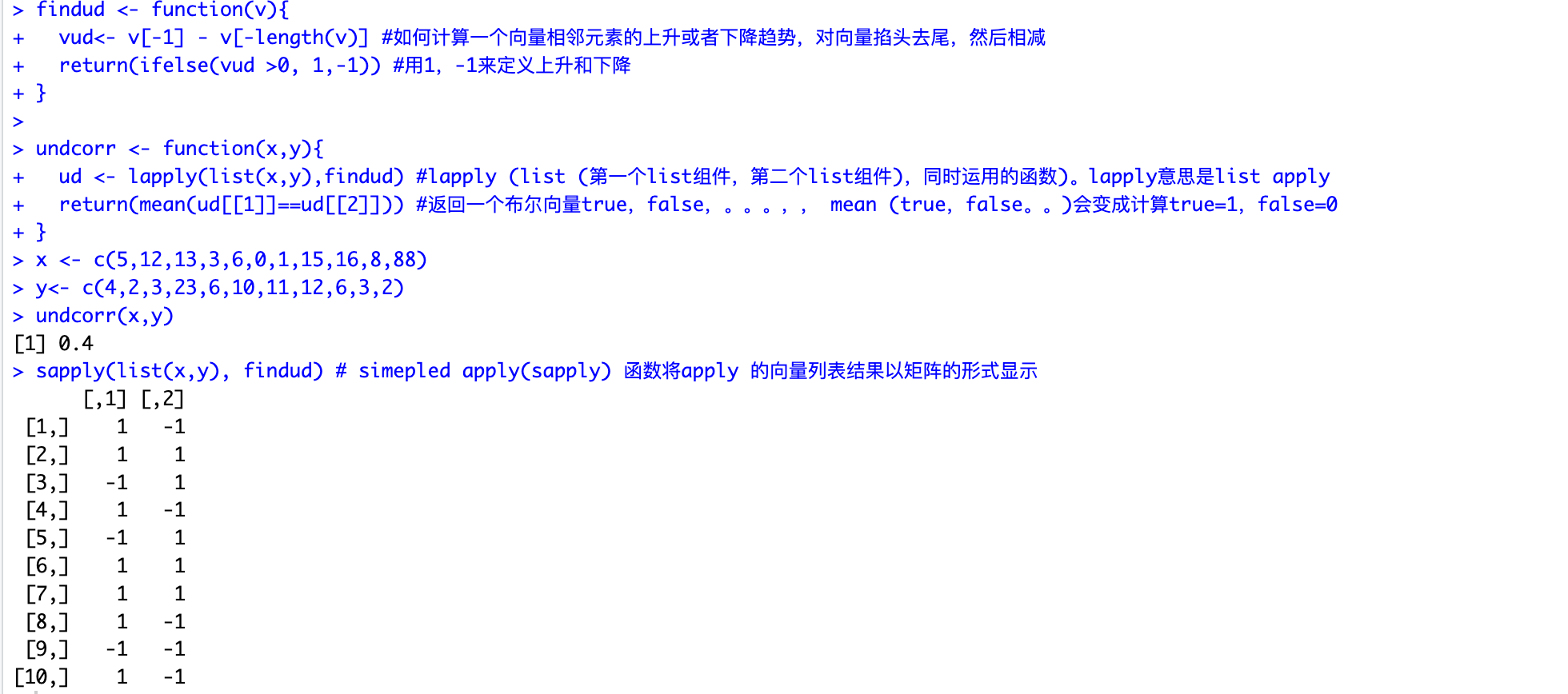
这种方式仅用于提取单个元素，而不能用于提取多个元素。

x[[1:3]] (这是错误的）

lapply（ ）函数对list强制执行所给的函数, sapply( ) 函数将lapply( ）的列表/向量结果变成结果变成向量/矩阵形式



ifelse ( ) , lapply（ ）, sapply( )函数的综合运用



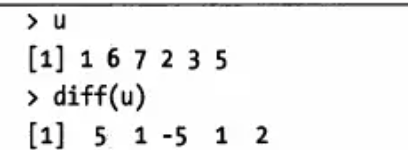
mean ( true, false, true, false .....) 会将true=1，false =0， 进行计算

比如：mean（ true, false, true, false）=mean（1，0，1，0）=2/4=0.4

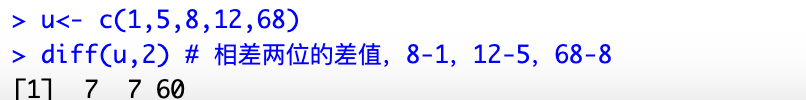
diff ( ) 函数用于计算一个数值向量中相邻元素的差异

方法一：ud <- v[-1] - v[-length(v)] 即创建两个新向量相减，第一个向量不要原向量的第一个元素，第二个向量不要原向量的最后一个元素

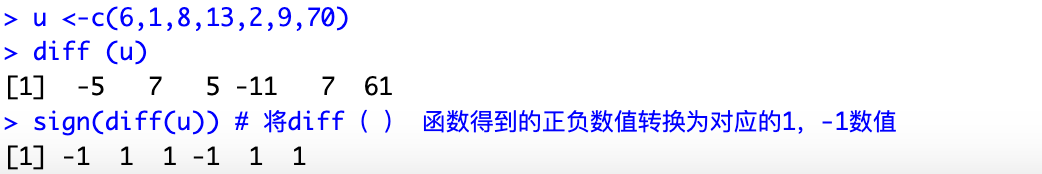
方法二：使用diff（）函数对向量做“滞后”运算，diff (所要计算的向量，相差几位）默认“滞后期”为1。



计算相差两位元素的差值



sign ( )函数用于将向量中的正负值转换为1，-1

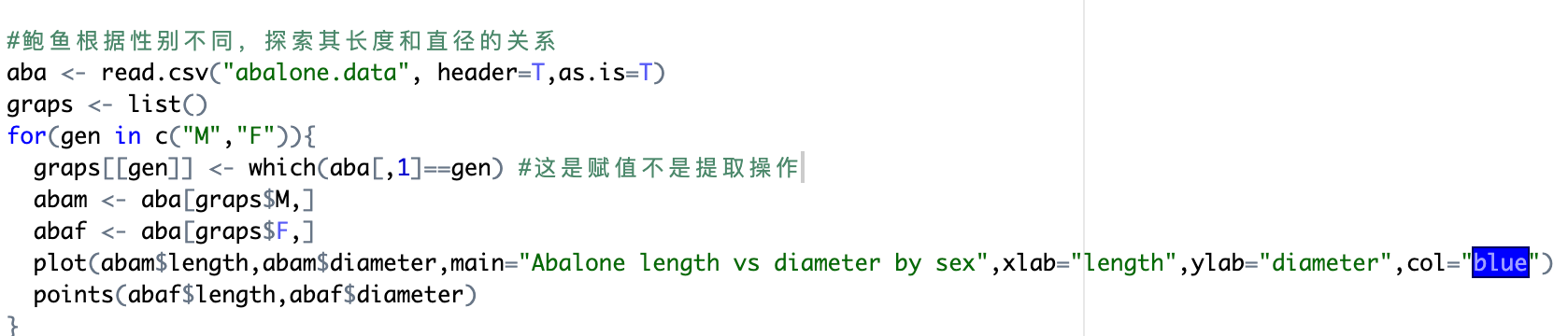


注意区分赋值与[[ ]] 的提取的操作

graps[ [gen] ]单独谈则是提取gen对应的元素出来，但是在下面代码的情况下则是将which的值赋予给graps[ [gen] ], 因为这行代码使用的的是赋值运算符, 也就是[ [ ] ] 在<-的左边则是赋值运算，在右边则是提取元素的操作

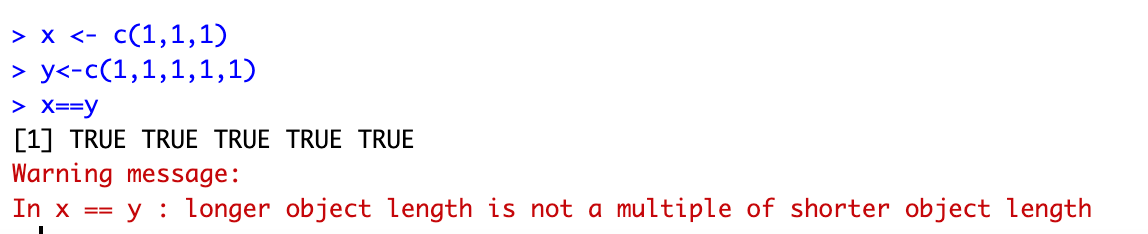
赋值操作：graps[[gen]]<- which(....)

提取操作：result <- graps[[gen]]

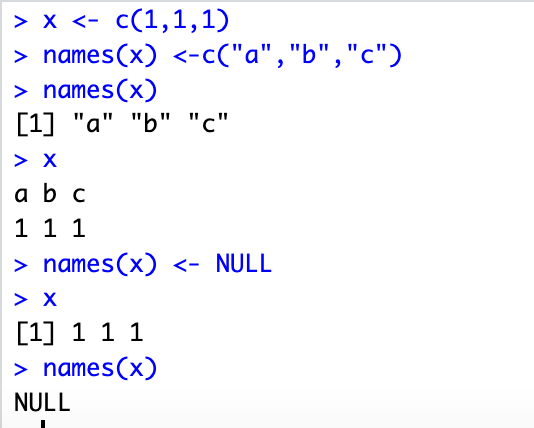


不能用 == 测试两个向量是否相等

因为如果两个向量的长度不一致， == 会使用循环补齐，得出和长向量相等长度的布尔向量，即使这个布尔向量的值都是true ，但这并不能表明这两个向量相等



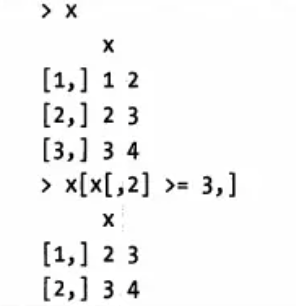
使用names（ ） 函数给向量命名

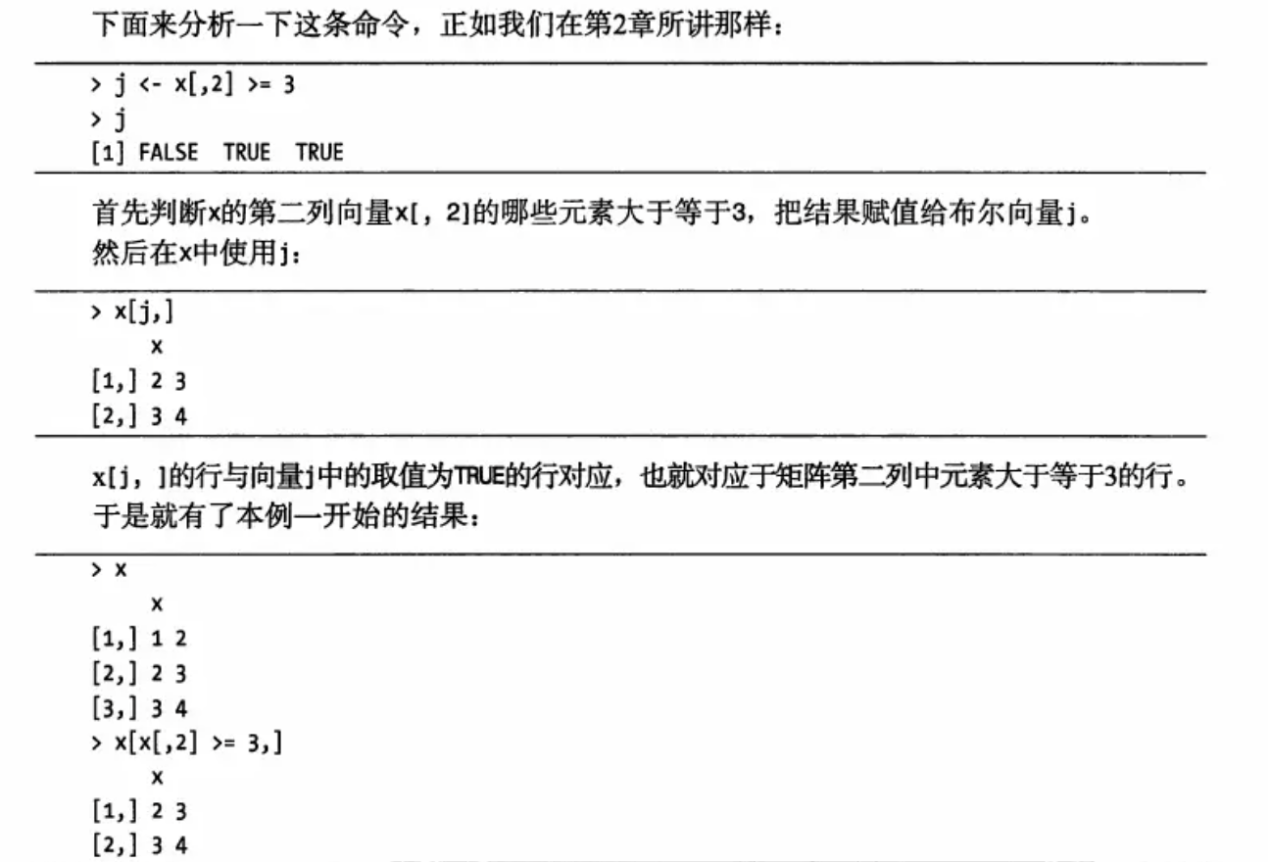


安装package时使用install 函数注意 加双引号“”install.packages("pixmap")

矩阵元素如何筛选

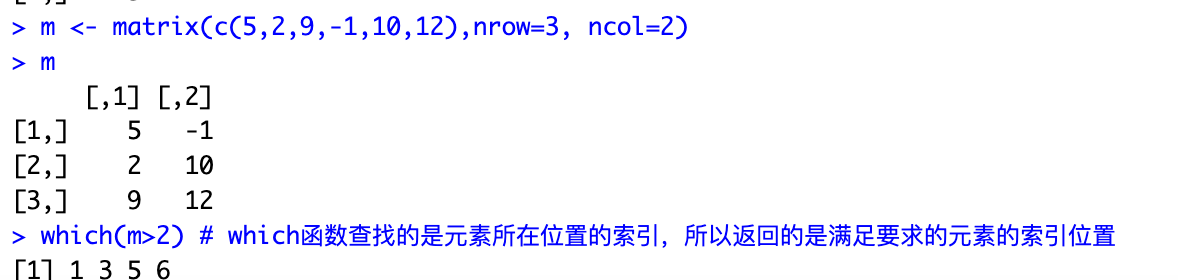
\*关键点： ==，>, <, >= 等运算得出的是布尔值向量，看要求，选出布尔向量对应的行/列为true的数值



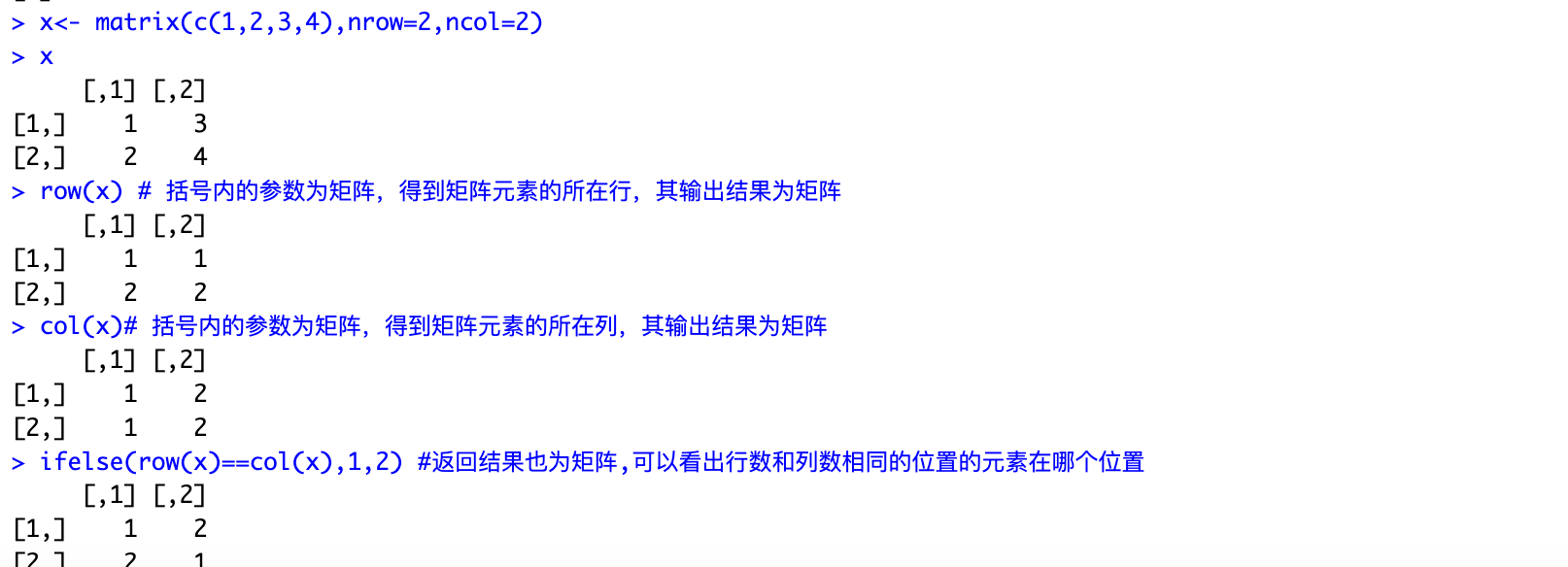


which函数在矩阵中如何使用

注意：矩阵中元素的索引位置是按照列额度顺序来的，因为矩阵的存储就是按照列来进行的。比如下列矩阵的第1，2，3，4个位置的元素对应的是5，2，9，-1



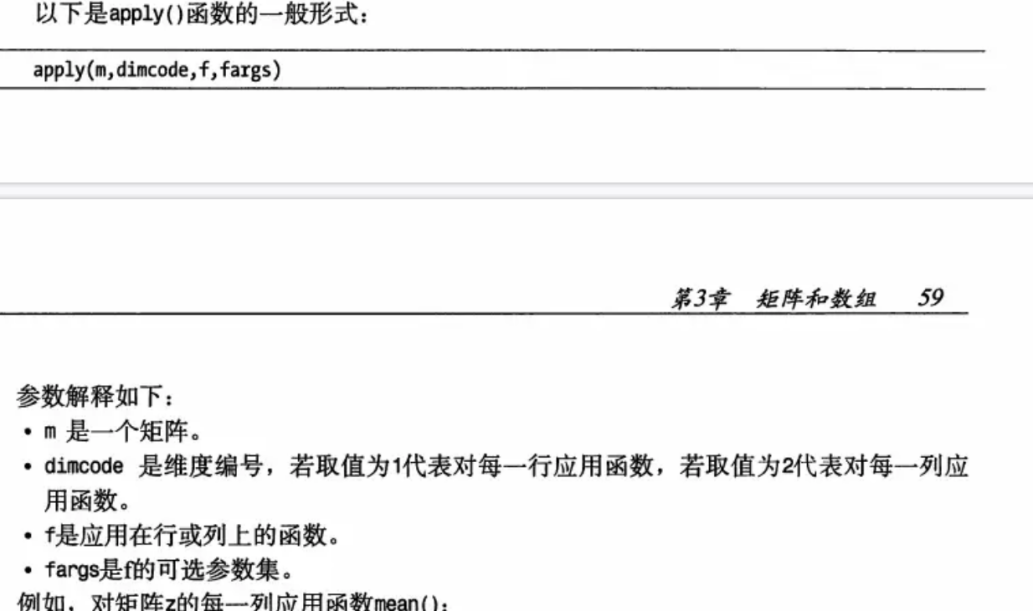
row( ) 和col ( ) 函数的使用，返回元矩阵元素对应的行或列号，返回值是矩阵

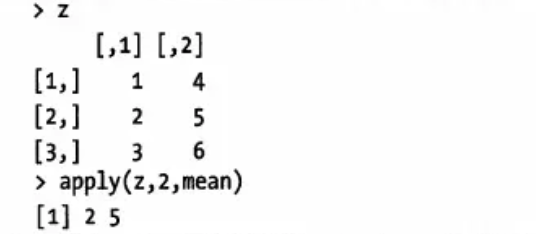


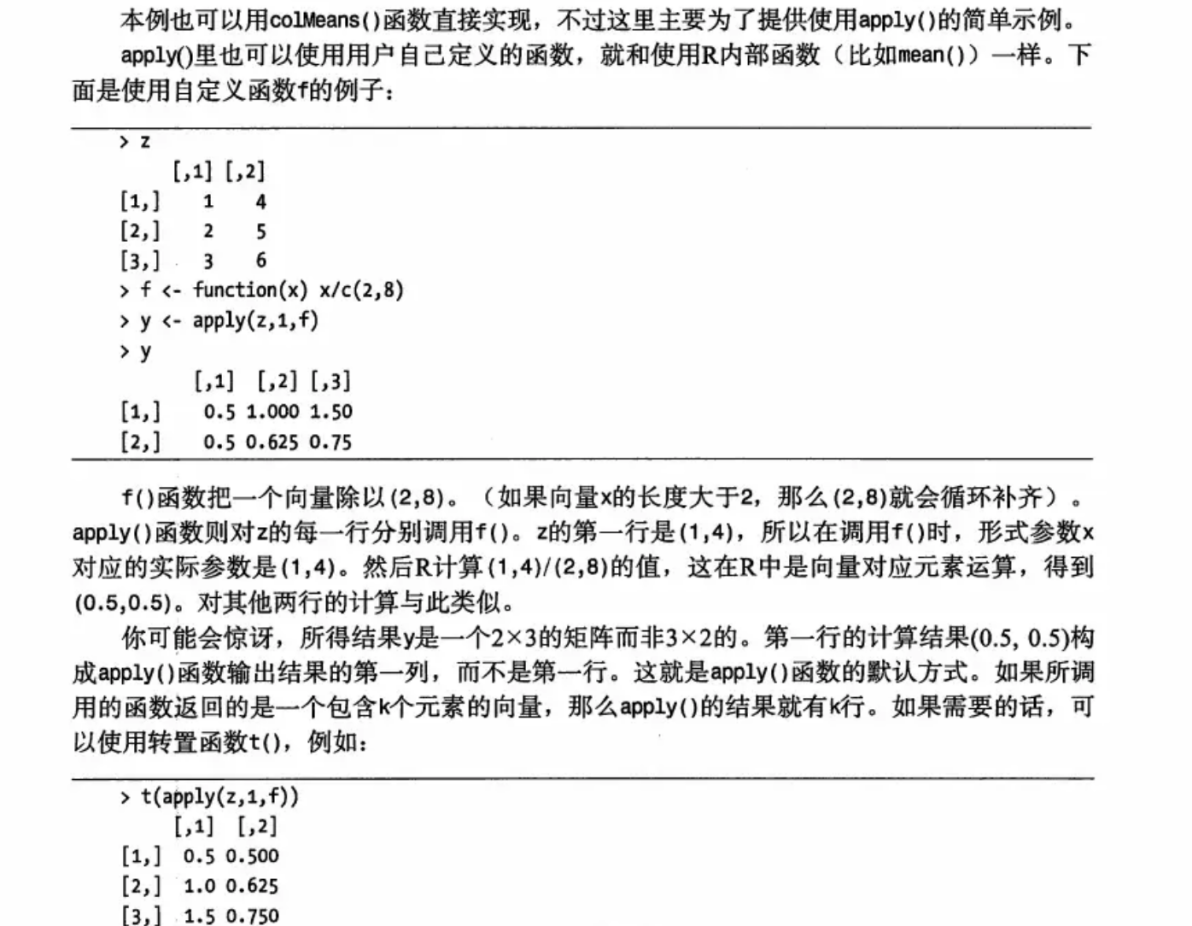
apply ( ) 函数的应用

以行或列为单位逐个元素进行计算，apply()函数最后的形式是向量矩阵取决于函数一次输出的元素数量，如果一次只输出1个元素，则最后形成一行的向量，如果单次输出结果为2个元素，则是2行的矩阵，同理，如果单次输出3个元素，则形成3行的矩阵。

apply( m,dimcode,f,fargs) 第四个参数其实是函数f内的参数。



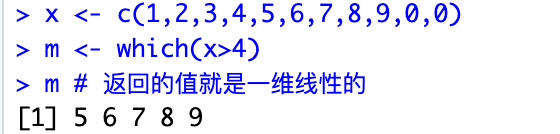




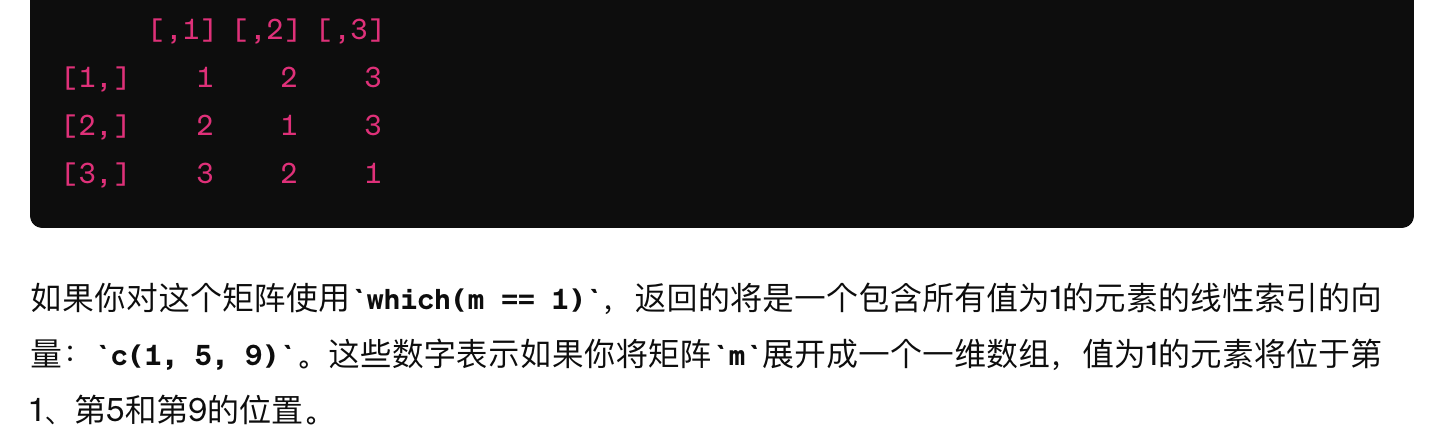
which ( ) 函数的对向量和矩阵应用的区别

which（）函数是用来寻找满足（ ）内条件元素的索引，返回结果都是一维线性的。

用于处理向量的是时候不用特别注意，因为向量的索引就是一维的比如下面：



但是在处理矩阵的时候需要注意，如果没有对which（） 函数设定参数进行说明那么其返回的索引结果也是一维的，这个索引的得到的顺序是按照列的来z字形， 比如下面

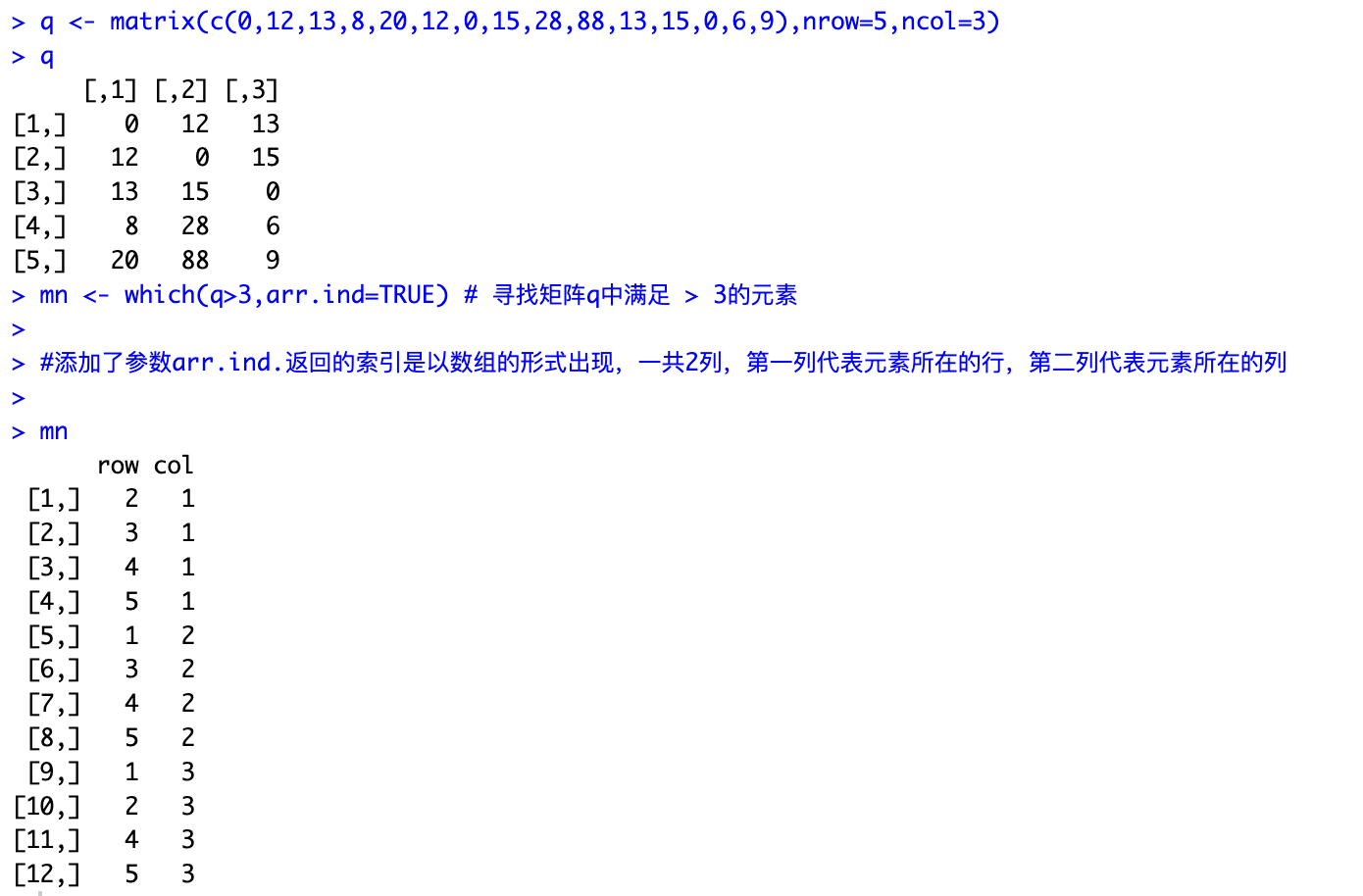


所以用which ( ) 函数处理矩阵的时候需要注意添加参数说明**arr.ind = TRUE .**

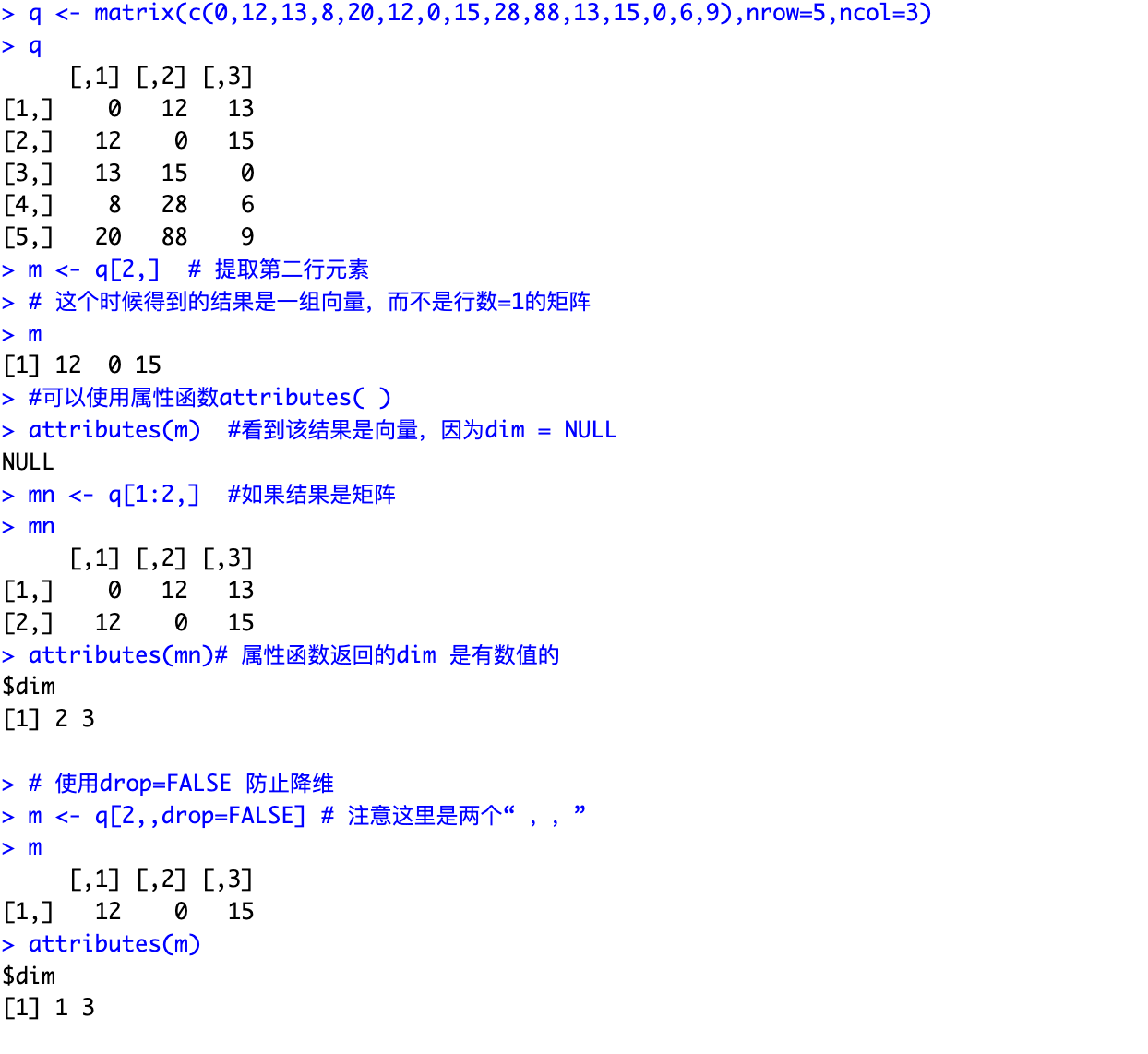
在R语言中，arr.ind是which函数的一个参数，用于控制函数返回值的格式。arr.ind参数的名称来源于“array indices”，意思是数组索引。当你在使用which函数查询满足某个条件的元素的位置时，arr.ind参数决定了返回的是线性索引还是数组（如矩阵）的维度索引。

当arr.ind = FALSE（默认值），which函数返回满足条件的元素的线性索引。这是一个整数向量，其中的每个数字代表满足条件的元素在将数组展平为一维数组后的位置。

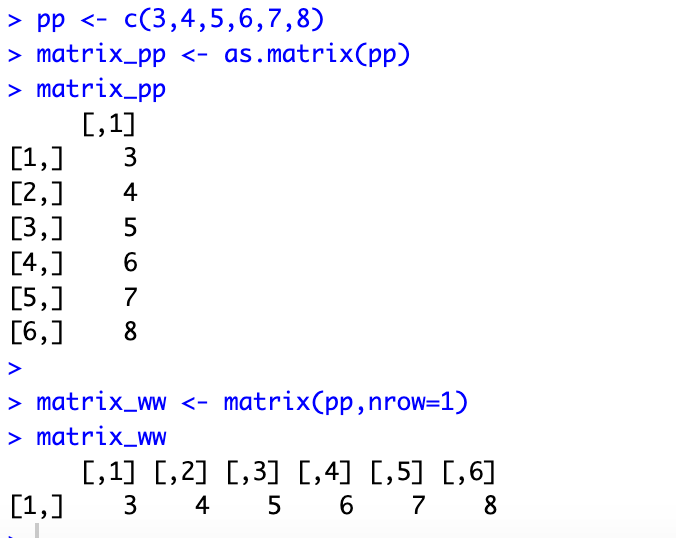
当arr.ind = TRUE时，which函数的行为改变。它不再返回一个线性索引的向量，而是返回一个矩阵，这个矩阵的每一行包含了一个满足条件的元素的多维索引。对于二维数组（矩阵），返回的矩阵有两列，分别代表行索引和列索引。



提取矩阵某一行/列的时候会导致得到的结果降维，使用drop=FALSE 参数避免降维，注意drop 前面一定是两个逗号“，， ”



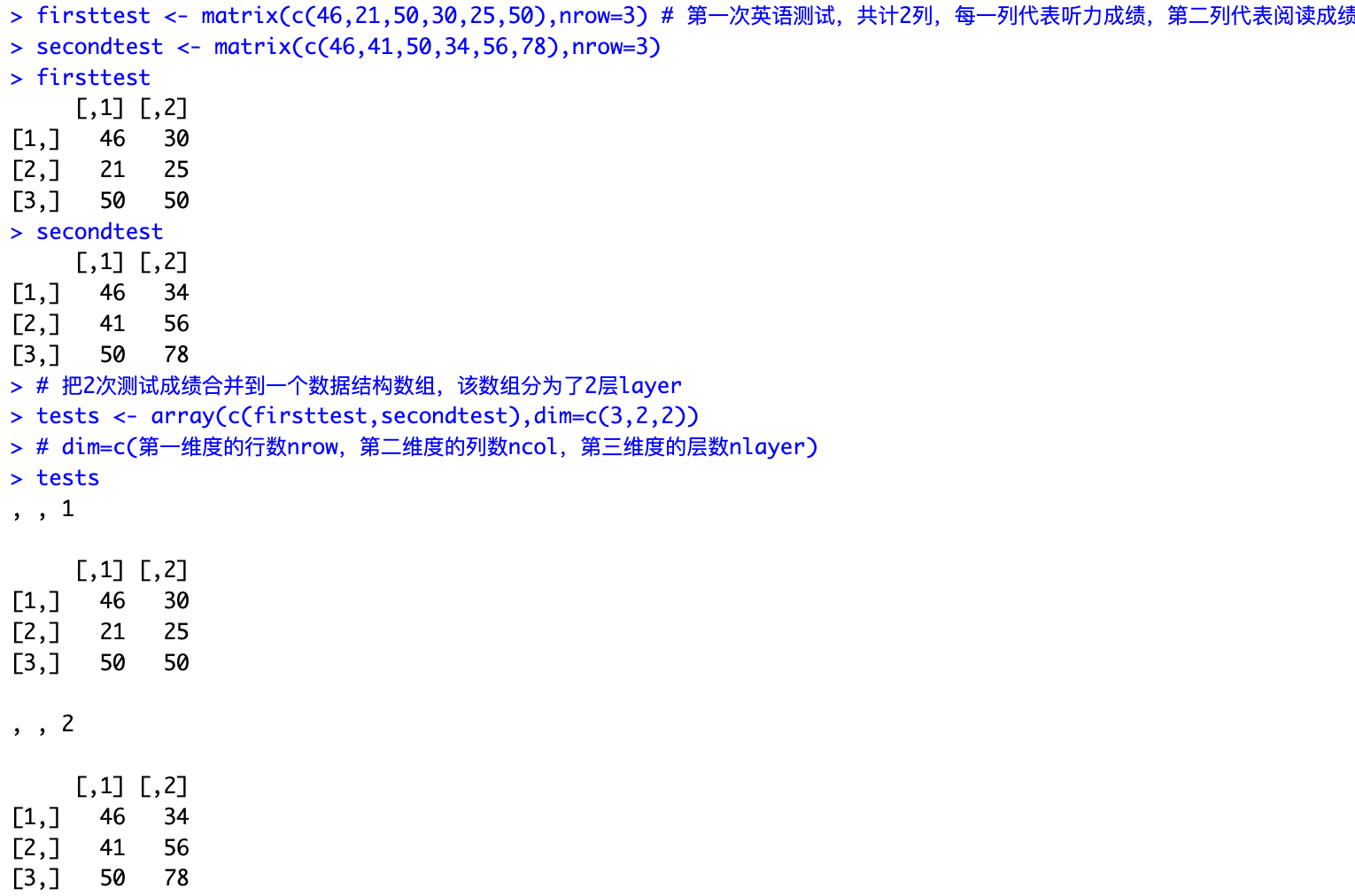
向量➡矩阵，使用as.matrix( ) 函数，注意这个转化得到的默认是创建一个只有一列的矩阵.如果你想将向量转换为只有一行的矩阵，可以通过matrix()函数来实现，指定nrow = 1或ncol = 1来控制矩阵的行数和列数：



高维数组把2个矩阵合并维三维数组，创立2层的数据。

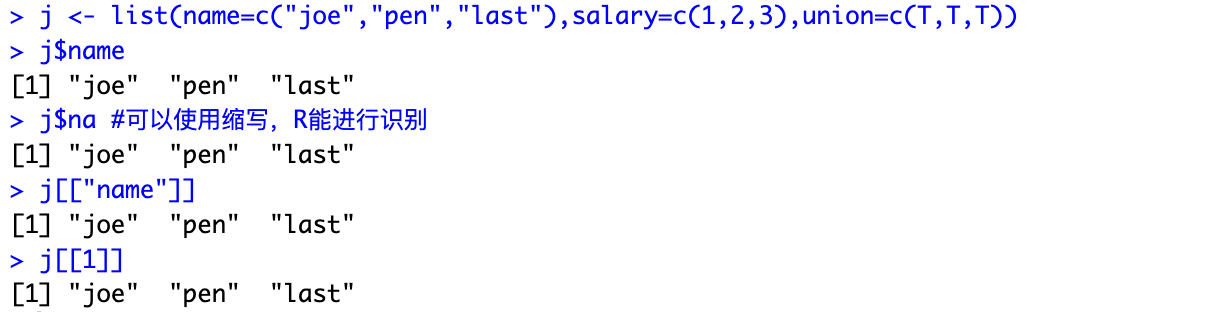
array ( data=c(矩阵1，矩阵2），dim=c( nrow, ncol, nlayer)

dim=c(第一维度的行数nrow，第二维度的列数ncol，第三维度的层数nlayer)



**列表**

list ( ) 列表索引的三种方式



list ( ) 列表添加组件，删除组件，修改组件（利用索引的格式）

修改列表组件时，并不要求确保组件中元素的数量与原来一致。列表是一种非常灵活的数据结构，它可以容纳不同类型和长度的元素。这意味着你可以随时更改列表中某个组件的内容，包括修改其长度。



names ( ) 函数只用在向量元素，列表的组件，不用在矩阵。矩阵使用rownames ( ) colnames ( ) 获得行和列的名字。

**元素/数值：**

（1）在向量和矩阵中，元素和它们的数值基本上是同义的，因为所有的元素都是相同类型的数值。

（2）在列表中，元素指的标签/组件，而下面的数据就是数值。

**atomic component**

在R语言的上下文中，当我们谈论“atomic component”（原子组件）时，通常是指构成复杂数据结构（如向量、矩阵或数据框）的最基本的数据元素。原子组件不是指列表的标签，而是指那些不可进一步分解的数据值本身，例如数值、字符、逻辑值等。

向量和矩阵的原子组件

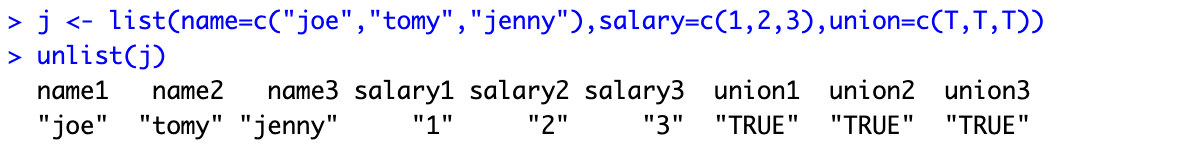
对于向量和矩阵，它们是由原子组件组成的，因为它们存储的数据元素都是相同类型的，且不能被进一步分解成更小的部分。比如，一个数值向量包含的每个数值就是一个原子组件。

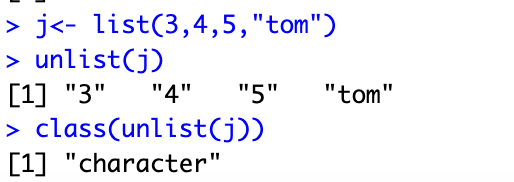
列表的原子组件

对于列表，情况稍微复杂一些。列表可以包含各种类型的数据，包括向量、矩阵、其他列表等。列表中的每个元素都可以是一个复杂的数据结构。但是，当我们讨论到列表或任何复杂数据结构中的“原子组件”时，我们是指那些构成这些复杂结构的最基本、不可再分的数据值，而不是指列表的标签。标签（或名字）只是用来引用或访问列表中的元素，它们本身不被视为数据的一部分。

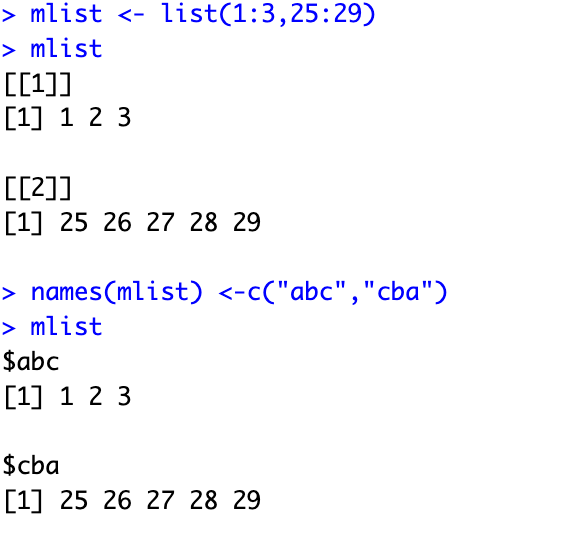
总之，“atomic component”指的是数据结构中那些最基本的、不可分割的数据元素，无论这些元素是数值、字符还是逻辑值等。这个概念有助于理解数据结构的构成和如何在R语言中进行有效的数据操作和分析。

unlist ( )函数，将列表从2维的数据解构为一维的向量，包含列表中所有的基础数据, 这个过程叫做去列表化，只要有可能，列表元素都会被强制转换为同一种存储格式，比如将数值转换为字符串。





给没有标签的列表命名

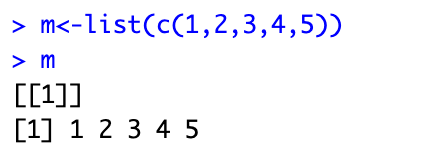


向量，矩阵，列表之间的相互转换

向量转换为矩阵：matrix(c(1,2,3,4,5) ,nrow=1), 或者使用as.matrix( 向量） 函数

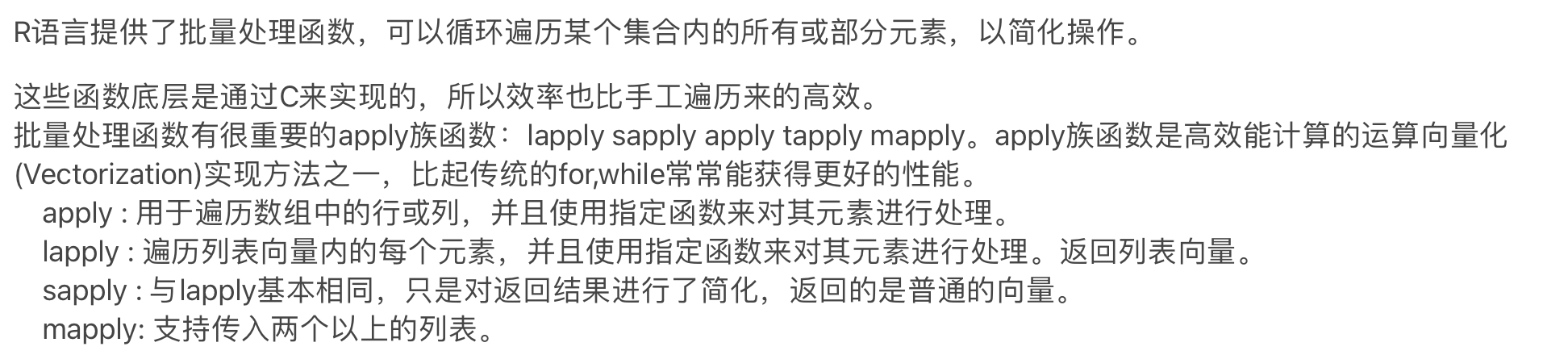
矩阵转换为向量：as.vector (矩阵）

向量转换为列表：list(c(1,2,3,4,5)),或者使用as.list( 向量） 函数



列表转换为向量：unlist( list(1:3,4:10), 函数）

apply系列函数在向量，矩阵，列表中的应用



数值排序函数sort ( ) 输出数值大小排序，order ( )函数输出数值大小的索引系

