R 语言基础

zhongyu1@genomics.cn

目录

一、 R 中的算数运算

1. 四则运算

在 R 中，加减乘除运算符分别用 +， -， *和 /来表示:

```
1+3-2*3
## [1] -2
5+14/2
## [1] 12
```

2. 整除运算

整除的商用%/%来进行获取；整除的取余数用%%来获取：

```
11 %/% 3 #整除
## [1] 3
11 %% 3 #取余
## [1] 2
```

3. 取整，保留小数和有效数字

取整运算：是将一个小数（浮点数）转化成整数。在 R 中取整运算相关的函数有 ceiling()，floor()，trunc()和 round()

```
## [1] 2
ceiling(3.14) #向上取整
## [1] 4
floor(3.14) #向下取整
## [1] 3
trunc(3.14) #向零取整
## [1] 3
trunc(-3.14)
## [1] -3
round(3.14) #默认取整
## [1] 3
```

有效小数：将一个小数近似为小数位特定的小数。在 R 中一般用 round()函数实现。在 R 中采用 banker 准则，即四舍六入五成双准则。“四舍”是指 ≤ 4 舍去；“六入”是指 ≥ 6 进一位；“五成双”分两种情况：(1)当 5 后面有数字时，舍 5 进一位。(2) 当 5 后面没有数字时，再分两种情况：(i)当 5 前一位为奇数时，舍 5 进一位；(ii)当 5 前一位为偶数时，舍 5 不进位。

```
pi #圆周率
## [1] 3.141593
round(pi, digits = 4) #保留4位小数
## [1] 3.1416
round(2.3) #四舍
## [1] 2
round(2.6) #六入
```

```
## [1] 3
round(2.50001) #当 5 后面有数字时，舍 5 进一位
## [1] 3
round(2.5) #当 5 后面没有数字时，5 前一位为奇数时，舍 5 进一位
## [1] 2
round(3.5) ##当 5 后面没有数字时，5 前一位为偶数时，舍 5 不进位
## [1] 4
```

有效数字：将一个数近似为特定有效位数的数字。R 中用 signif()函数实现。

```
signif(pi) #默认保留 6 位有效数字
## [1] 3.14159
signif(pi, 4) #保留 4 位有效数字
## [1] 3.142
```

4. 幂指数运算

幂指数运算符用^来表示，自然指数运算使用 exp()函数

```
3^2
## [1] 9
exp(2) #自然指数
## [1] 7.389056
```

5. 对数运算

对数运算常用相关函数有 log(), log2(), log10()

```
log(2) #以自然常数 e 为底的对数
## [1] 0.6931472
log10(100) #以 10 为底的对数
## [1] 2
log2(1024) #以 2 为底的对数
## [1] 10
log(x = 81, base = 9) #以 9 为底的对数
## [1] 2
```

6. 绝对值

绝对值运算用 abs()函数

```
abs(1)
## [1] 1
abs(-1)
## [1] 1
```

7. 平方根

平方根运算用 sqrt()函数

```
sqrt(100)
## [1] 10
```

思考：如何计算立方根？

8. 三角函数运算

R 中的相关三角函数 sin()，cos()，tan()分别表示正弦、余弦、正切。

```
pi #圆周率
## [1] 3.141593
sin(pi/6) #正弦
## [1] 0.5
cos(pi/3) #余弦
## [1] 0.5
tan(pi/4) #正切
## [1] 1
```

二、 R 中的关系运算

在 R 中，包含逻辑常量 TRUE 和 FALSE 分别表示逻辑常量真和假，其值分别为 1 和 0。

R 语言中，>，<，>=，<=，== 和 != 运算符分别表示大于，小于，大于等于，小于等于，等于和不等于。

```
2 > 1
## [1] TRUE
2 < 1
## [1] FALSE
2 >= 2
## [1] TRUE
2 <= 1
## [1] FALSE
2 == 1
## [1] FALSE
2 != 1
## [1] TRUE
T == 1
## [1] TRUE
```

三、 R 中的逻辑运算

&，|，和 ！运算符分别表示逻辑与，或和非。

```
TRUE & FALSE
## [1] FALSE
TRUE & TRUE
## [1] TRUE
TRUE | FALSE
## [1] TRUE
FALSE | FALSE
## [1] FALSE
```

```
!FALSE
## [1] TRUE
(2 > 1) & (3 <= 1)
## [1] FALSE
```

四、 R 中的变量与赋值

在 R 语言中，使用变量前不需要声明变量的类型。变量是在赋值运算后才创建的。

变量命名规则：1. 变量名区分大小写；变量名的首字母不能是数字和符号；变量名不能包含空格。

在 R 中，用赋值运算符<-来进行赋值运算，赋值运算符<-之间不能有空格。在 R 中，常用的是左赋值运算符<-，其实也可以用右赋值运算符->（意味着将左边的值赋给右边的变量）。等号=也可以进行赋值运算，但是容易与函数中参数赋值混淆，尽量避免用=来进行变量赋值。

```
_x <- 1
Error: unexpected input in "_"
x <- 1
x
## [1] 1
X <- 2
X
## [1] 2
y <- 2
y
## [1] 2
z <- y - x
z
## [1] 1
my name <- 'zhongyu'
Error: unexpected symbol in "my name"
my_name <- 'zhongyu'
100 -> a -> b -> c
a
## [1] 100
b
## [1] 100
c
## [1] 100
```

查看及删除变量

```
ls() #列出环境中的所有变量
## [1] "a"       "b"       "c"       "my_name" "x"       "X"       "y"
## [8] "z"
```

```
rm(a) #清除变量a
ls()
## [1] "b"       "c"       "my_name" "x"       "X"       "y"       "z"
rm(list = ls()) #清除所有变量
ls()
## character(0)
```

五、 数据类型及数据结构

1. 基本数据类型

R 语言的基本数据类型包括**数值型**、**字符型**、**逻辑型**、**因子**和**特殊值**：

1) 数值型（numeric，integer, complex）

```
x <- 1
x
## [1] 1
class(x)
## [1] "numeric"
x <- 1L # x <- as.integer(1)
x
## [1] 1
class(x)
## [1] "integer"
x <- 1+1i
x
## [1] 1+1i
class(x)
## [1] "complex"
```

2) 字符型（character）

```
x <- "1"
x
## [1] "1"
class(x)
## [1] "character"
```

3) 逻辑型（logical）

```
x <- TRUE # x <- as.logical(1)
x
## [1] TRUE
class(x)
## [1] "logical"
```

4) 因子（factor）

```
x <- factor("1")
x
## [1] 1
## Levels: 1
class(x)
## [1] "factor"
```

5) 特殊值：NA（缺失值）、Inf（无穷大）、-Inf（无穷小）、NaN（非数值）、NULL（空值）

NA 表示缺失值，这在我们读取一些数据的时候可能会遇到。在 R 中可用 is.na()函数来判断是否为缺失值：

```
x <- c(1,NA,3,4,NA)
is.na(x)
## [1] FALSE  TRUE FALSE FALSE  TRUE
```

需要删除缺失值或统计有多少个缺失值，可以通过下面代码来实现：

```
x[!is.na(x)]     ## 删除缺失值
## [1] 1 3 4
sum(is.na(x))    ## 统计缺失值的个数
## [1] 2
```

Inf/-Inf 表示无穷大和无穷小，在 R 中可用 is.finite() 或 is.infinite()来判断是否为无穷大/小数：

```
1/0
## [1] Inf
-1/0
## [1] -Inf
is.finite(1/0)
## [1] FALSE
is.infinite(1/0)
## [1] TRUE
```

NaN 表示非数值，某些运算会导致结果为非数值。在 R 中，用 is.nan()来判断是否为非数值，

```
0/0
## [1] NaN
Inf-Inf
## [1] NaN
is.nan(0/0)
## [1] TRUE
```

NULL 表示空值，表示没有内容，其长度为 0。常见于函数参数或初始化变量。

```
x <- NULL
length(x)
## [1] 0
is.null(x)
```

```
## [1] TRUE
```

2. 数据结构

1) 向量（vector）

向量是 R 语言最基本的数据类型。向量中只能存放同一类型的数据，常见的向量类型有**数值型向量**、**字符型向量**和**逻辑型向量**。

创建向量

在 R 中，使用 c()函数创建向量是最常用的方式，此外，还可以用 vector()函数进行创建，常用于创建指定类型或长度的空向量。

```
c(1,2,3,4)
## [1] 1 2 3 4
1:4
## [1] 1 2 3 4
c("a","b","c")
## [1] "a" "b" "c"
c("a","b",1,2,TRUE)
## [1] "a"    "b"    "1"    "2"    "TRUE"
x <- vector(mode = "character", length = 10) #创建一个长度为 10 的字符型向量
```

当 c()函数中元素类型型不同时，会强制转化成同一类型；而且每种类型的强弱不一样，强弱顺序为：character > complex > numeric > integer > logical。

向量的长度

向量的长度是指向量中元素的个数，可以用 length()函数来查看：

```
x <- vector(mode = "character", length = 10) #创建一个长度为 10 的字符型向量
length(x)
## [1] 10
```

访问及修改向量中的元素

在 R 中，[]来指定索引来访问向量，索引从位置 1 开始。

```
x <- 1:4
x[2] #获取向量 x 的第 2 个元素
## [1] 2
x[c(2,4)] #获取向量 x 的第 2、4 个元素
## [1] 2 4
x[-2] #访问除第 2 个元素外的其它元素
## [1] 1 3 4
```

可以通过赋值运算来改变向量中的内容：

```
x[2] <- 5
## [1] 1 5 3 4
```

创建有名称向量

可用 c()函数直接创建有名称向量，或用 names()函数给无名向量的每个元素命名。

```
x_wtih_name <- c(a=1, b=2, c=3, d=4)
x_wtih_name
## a b c d
## 1 2 3 4
names(x) <- c("a","b","c","d")
x
## a b c d
## 1 5 3 4
names(x) #输出向量名称
## [1] "a" "b" "c" "d"
```

对于有名称向量，通过指标和名称都可以访问元素，分四种情况：

x[i]：返回向量元素的值和名称
x[[i]]：只返回向量元素的值
x[元素名称]：返回向量元素的值和名称
x[[元素名称]]：只返回向量元素的值

```
x[2]
## b
## 5
x[[2]]
## [1] 5
x["b"]
## b
## 5
x[["b"]]
## [1] 5
```

向量的基本运算

向量间的运算为对应元素间的运算：

```
x <- c(1, 2, 3)
y <- c(4, 5, 6)
x+y
## [1] 5 7 9
x*y
## [1]  4 10 18
```

有关向量的函数

- max(x)：找出 x 向量中的**最大值**
- min(x)：找出 x 向量中的**最小值**
- range(x)：找出 x 向量中的**最小值和最大值**
- length(x)：求 x 向量的**长度**
- sum(x)：求 x 向量的**元素和**
- prod(x)：求 x 向量的**元素积**
- mean(x)：求 x 向量的**算术平均值**
- median(x)：求 x 向量的**中位数**
- var(x)：求 x 向量的**方差**
- sd(x)：求 x 向量的**标准差**
- cor(x, y)：求 x 向量和 y 向量的**相关系数**
- sort(x)：将 x 向量**按大小顺序排序**
- rank(x)：返回 x 向量中**对应元素的排名**
- order(x)：返回对应“排名”的元素在 x 向量中的**位置**
- quantile(x)：返回 x 向量的**最小值**、**下分位数**、**中位数**、**上分位数**和**最大值**
- table(x)：返回 x 向量的**元素统计频数**
- unique(x)：**删除** x 向量中的**重复元素**
- union(x, y)：x 向量和 y 向量的**并集**
- intersect(x, y)：x 向量和 y 向量的**交集**
- setdiff(x, y)：x 向量和 y 向量的**差集**（即返回属于 x 向量的元素但不属于 y 向量）
- x %in% y：判断 x 向量中的元素是否在 y 向量中

2） 矩阵（matix）

矩阵可以理解为向量+维度属性(nrow, ncol)，维度属性为矩阵的行数和列数。

创建矩阵

在 R 中，使用 matrix()函数来创建矩阵是最常用的方式。matrix()的原型为：matrix(data=NA, nrow=1, ncol = 1, byrow=FALSE, dimnames=NULL)，其中参数的意义分别为：

data：包含了矩阵的元素，一般是个向量，默认情况下是 NA；

nrow 和 ncol：设定矩阵的行、列数目；一般这两个值只需设定一个，另外一个值可根据元素个数自动给出；

byrow：设定矩阵是按行(byrow=TRUE)填充还是按列(byrow=FALSE)填充，默认情况下按列填充；

dimnames：包含了以字符型向量表示的行名和列名，是一个列表，默认情况下没有行列名数组。

```
x <- matrix(1:6, nrow=3, ncol = 2)
```

```
x
##      [,1] [,2]
## [1,]    1    4
```

```
## [2,]    2    5
## [3,]    3    6
x <- matrix(1:6, nrow=3, ncol = 2, byrow = TRUE) #按列进行填充
x
##      [,1] [,2]
## [1,]    1    2
## [2,]    3    4
## [3,]    5    6
```

可用 dim()函数读取矩阵的维度属性

```
dim(x)
## [1] 3 2
```

给矩阵添加行名和列名

```
rnames <- c('R1', 'R2', 'R3')   ##行名
cnames <- c('C1', 'C2')  ##列名
x <- matrix(1:6, nrow=3, ncol = 2, byrow = TRUE, dimnames =
list(rnames, cnames)) #通过 dimnames 参数添加行列名
x
##    C1 C2
## R1  1  2
## R2  3  4
## R3  5  6
```

也可通过使用 rownames()、colnames()函数来给矩阵添加行、列名

```
rownames(x) <- c('r1', 'r2', 'r3')   ##行名
colnames(x) <- c('c1', 'c2')  ##列名
x
##    c1 c2
## r1  1  2
## r2  3  4
## r3  5  6
```

此外，也可通过使用 dim()函数来创建矩阵，其原理是通过改变维度使向量变为矩阵。t()函数可用于矩阵的转置。

```
y <- 1:6
dim(y) <- c(3,2) #3 行 2 列
y
##      [,1] [,2]
## [1,]    1    4
## [2,]    2    5
## [3,]    3    6
t(y)
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
```

访问及修改矩阵的元素

与向量元素的访问类似，只是相对于向量增加了一个维度。

```
x
##    c1 c2
## r1  1  2
## r2  3  4
## r3  5  6
x[1,2] #访问第 1 行第 2 列元素
## [1] 2
x[2,] #访问第 2 行元素,返回向量
## c1 c2
##  3  4
x[, 2]  #访问第 2 列元素,返回向量
## r1 r2 r3
##  2  4  6
x[c(1, 3), c(1, 2)]  #访问第 1、3 行第 1、2 列元素，返回为矩阵
##    c1 c2
## r1  1  2
## r3  5  6
### 也可通过行、列名来访问元素
x['r1', 'c2']
## [1] 2
x['r2', ]
## c1 c2
##  3  4
x[c('r1','r3'), c('c1','c2')]
##    c1 c2
## r1  1  2
## r3  5  6
```

修改矩阵元素也与向量类似，通过赋值进行操作。

```
x[1, 2] <- 0 #将第 1 行第 2 列元素改为 0
x
##    c1 c2
## r1  1  0
## r2  3  4
## r3  5  6
x[1, ] <- 0 #将第 1 行元素都改为 0
x
```

```
##    c1 c2
## r1  0  0
## r2  3  4
## r3  5  6
x[-1, ] #删掉第1行
##    c1 c2
## r2  3  4
## r3  5  6
```

通过 rbind()和 cbind()函数可对矩阵添加行和列。

```
cbind(x,c(1,1,1)) #在原矩阵x后面添加一列
##    c1 c2
## r1  0  0 1
## r2  3  4 1
## r3  5  6 1
rbind(x,r4=c(1,1)) #在原矩阵x后面添加一行
##    c1 c2
## r1  0  0
## r2  3  4
## r3  5  6
## r4  1  1
a <- cbind(x,x) #按列拼接矩阵
a
##    c1 c2 c1 c2
## r1  0  0  0  0
## r2  3  4  3  4
## r3  5  6  5  6
```

矩阵的行、列计算

```
x <- matrix(rnorm(16), ncol=4) #rnorm()函数用于生成长度为n的正态分布随机数构成的向量
```

```
x
##           [,1]       [,2]       [,3]       [,4]
## [1,] -0.7109287 -0.9484410 -1.3108288  0.5369116
## [2,]  1.9655609 -0.1685957  0.3635666 -0.9471064
## [3,]  0.3007380  0.4640173  0.2405610 -0.1517716
## [4,] -1.1308096 -2.3560762 -0.1373174  0.8038023
mean(x[1,]) #求第1行的平均值
## [1] -0.6083217
sd(x[1,]) #求第1行的标准差
## [1] 0.8023476
rowSums(x) #行和
## [1] -2.4332869  1.2134255  0.8535447 -2.8204008
colSums(x) #列和
```

```
## [1]  0.4245606 -3.0090956 -0.8440186  0.2418360
rowMeans(x) #行均值
## [1] -0.6083217  0.3033564  0.2133862 -0.7051002
colMeans(x) #列均值
## [1]  0.10614016 -0.75227390 -0.21100464  0.06045899
```

如要计算每行的标准差或者最大值，可以使用 apply()函数来实现。apply()函数的原型为 apply(X, MARGIN, FUN, ...)，其中：X 为矩阵或数组；MARGIN 用来指定是对行运算还是对列运算，MARGIN = 1 表示对行运算，MARGIN = 2 表示对列运算；FUN 用来指定运算函数；... 用来指定 FUN 中需要的其它参数。

```
apply(x, 1, sum) #行和
## [1] -0.9929037 -0.4175454 -3.1717901  2.5948096
apply(x, 2, mean) #列均值
## [1]  0.18783242 -0.60858013 -0.05699559 -0.01911408
apply(x, 1, sd) #行标准差
## [1] 0.5626507 1.1639978 0.4696055 0.8713161
apply(x, 1, max) #每行最大值
## [1]  0.4791384  1.1755865 -0.2560729  1.8506358
x
##           [,1]       [,2]       [,3]       [,4]
## [1,] -0.8931653  0.4791384 -0.3151772 -0.2636995
## [2,]  0.3360308 -1.5779753 -0.3511875  1.1755865
## [3,] -0.5421716 -1.1983594 -0.2560729 -1.1751862
## [4,]  1.8506358 -0.1371243  0.6944553  0.1868428
apply(x, 1, function(x) x+1) #自定义函数
##          [,1]      [,2]      [,3]     [,4]
## [1,] 0.1068347  1.3360308  0.4578284 2.8506358
## [2,] 1.4791384 -0.5779753 -0.1983594 0.8628757
## [3,] 0.6848228  0.6488125  0.7439271 1.6944553
## [4,] 0.7363005  2.1755865 -0.1751862 1.1868428
```

3) 数组（array）

数组与矩阵类似，但是维度可以大于 2。向量是一维数组，矩阵是二维数组。与向量和矩阵一样，数组的元素必须也是同一类型的数据。

创建数组

使用 array()函数来创建数组。array()函数默认为：array(data = NA, dim = length(data), dimnames = NULL)，其中：data 给定数组数据，默认情况下是 NA；dim 用来指定数组的维度，默认情况下是一维数组；dimnames 设定各维度的名称，必须是个列表，默认情况下无名称。

```
x <- array(data = 1:24, dim = c(4,6)) #4x6 的 2 维数组，元素为 1 到 24 的序列，即矩阵。

x
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    1    5    9   13   17   21
## [2,]    2    6   10   14   18   22
## [3,]    3    7   11   15   19   23
## [4,]    4    8   12   16   20   24
x <- array(data = 1:24, dim = 2:4)  # 2x3x4 的三维数组，元素为 1 到 24 的序列
x
## , , 1
##
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
##
## , , 2
##
##      [,1] [,2] [,3]
## [1,]    7    9   11
## [2,]    8   10   12
##
## , , 3
##
##      [,1] [,2] [,3]
## [1,]   13   15   17
## [2,]   14   16   18
##
## , , 4
##
##      [,1] [,2] [,3]
## [1,]   19   21   23
## [2,]   20   22   24
dim(x) #数组维度
## [1] 2 3 4
## 添加各维度名称
dimnames(x)[[1]] <- c("male","female")
dimnames(x)[[2]] <- c("young","middle","old")
dimnames(x)[[3]] <- c("A","B","C","D")
x
## , , A
##
##        young middle old
## male       1      3   5
## female     2      4   6
##
```

```
## , , B
##
##        young middle old
## male       7      9  11
## female     8     10  12
##
## , , C
##
##        young middle old
## male      13     15  17
## female    14     16  18
##
## , , D
##
##        young middle old
## male      19     21  23
## female    20     22  24
```

矩阵这一数据类型较为不常用仅作了解。

4) 数据框（data.frame）

数据框主要用于存储表格数据，结构与矩阵类似，与矩阵不同的是元素类型可以不同。

创建数据框

```
df <- data.frame(id=c(1:4),name=c("a","b","c","d"),male=c(T,F,T,F))
df
##   id name  male
## 1  1    a  TRUE
## 2  2    b FALSE
## 3  3    c  TRUE
## 4  4    d FALSE
dim(df) #维度
## [1] 4 3
nrow(df) #行数
## [1] 4
ncol(df) #列数
## [1] 3
```

数据框元素访问

```
df[1,] #访问第一行
##   id name male
## 1  1    a TRUE
df[,2] #访问第二列
## [1] a b c d
## Levels: a b c d
```

```
df[,"id"] #访问列名为"id"列
## [1] 1 2 3 4
```

除了可用[]的方式访问，还可以用$对数据框元素进行访问。

```
df <-
data.frame(id=c(1:4),name=c("a","b","c","d"),male=c(T,F,T,F),strings
AsFactors = F) # stringsAsFactors = F，不将字符串作为因子
df$name #访问列名为"name"列
## [1] "a" "b" "c" "d"
```

获取数据框的子集

```
df[df$male==T,] #获取男性的数据
##   id name male
## 1  1    a TRUE
## 3  3    c TRUE
subset(df, male==T) #通过subset()函数获取子集
##   id name male
## 1  1    a TRUE
## 3  3    c TRUE
```

添加列行及合并数据框。

```
age <- data.frame(id=c(1:4),age=(20:23))
cbind(df,age)
##   id name  male id age
## 1  1    a  TRUE  1  20
## 2  2    b FALSE  2  21
## 3  3    c  TRUE  3  22
## 4  4    d FALSE  4  23
data.frame(df,age)
##   id name  male id.1 age
## 1  1    a  TRUE    1  20
## 2  2    b FALSE    2  21
## 3  3    c  TRUE    3  22
## 4  4    d FALSE    4  23
merge(df,age)
##   id name  male age
## 1  1    a  TRUE  20
## 2  2    b FALSE  21
## 3  3    c  TRUE  22
## 4  4    d FALSE  23
```

5) 列表（list）

列表可以包含不同类型的对象，大多数函数返回的对象都是以列表的形式进行存储。

创建列表

R 中使用 list()函数来创建列表

```
l1 <- list(id=c(1:4),name=c("a","b","c","d"),male=c(T,F,T,F),
x=1+1i)
l1
## $id
## [1] 1 2 3 4
##
## $name
## [1] "a" "b" "c" "d"
##
## $male
## [1]  TRUE FALSE  TRUE FALSE
##
## $x
## [1] 1+1i
```

str()函数来获取 list 内部结构。

```
str(l1)
## List of 4
##  $ id  : int [1:4] 1 2 3 4
##  $ name: chr [1:4] "a" "b" "c" "d"
##  $ male: logi [1:4] TRUE FALSE TRUE FALSE
##  $ x   : cplx 1+1i
```

列表中元素访问

```
l1[1] #通过数字来索引，[]操作符返回列表
## $id
## [1] 1 2 3 4
l1[[1]] #通过数字来索引，[[]]操作符返回元素内容
## [1] 1 2 3 4
l1["name"] #通过字符串索引，与[[]]类似
## $name
## [1] "a" "b" "c" "d"
l1$male #通过$操作索引
## [1]  TRUE FALSE  TRUE FALSE
```

六、 函数

R 中的变量包括数据和函数，R 有很多内置的函数，之前所提到的基本上都是 R 内置的函数，例如，sum()，mean()，sd()等。函数的结构包括函数名，输入，函数体和输出。R 中使用 function()定义函数。

创建简单的函数——平方函数

```
square <- function(x){
  return(x^2)
}
square(2)
## [1] 4
square <- function(x, power=2){
  return(x^power)
}
square(2) #默认为平方
## [1] 4
square(2, 3) #power=3，返回3次方
## [1] 8
```

七、 循环和控制流

R 中常用的循环有 for 循环和 while 循环。for 循环对集合中每一个元素执行相同的操作。while 循环需要在条件判断为真时，循环体被执行。

```
for(i in 1:3){
  cat(i, "+", 1, "=", i+1, "\n")
}
## 1 + 1 = 2
## 2 + 1 = 3
## 3 + 1 = 4
i <- 0
while(i < 3){
  print(i)
  i <- i+1
}
## [1] 0
## [1] 1
## [1] 2
```

控制流：if 和 if else 语句

if 判断为真时，if 后面的表达式将被执行，为假时则不执行。if else 语句不是选不选择执行相应的表达式，而是选择不同的表达式进行执行，如果判断为真时，if 后面的表达式将被执行，否则执行 else 后面的表达式。

```
for(i in 1:3){
  if(i < 3){
    cat(i, "+", 1, "=", i+1, "\n")
  }
}
## 1 + 1 = 2
## 2 + 1 = 3
```

```
for(i in 1:3){
  if(i < 3){
    cat(i, "+", 1, "=", i+1, "\n")
  }else{
    print(i)
  }
}
## 1 + 1 = 2
## 2 + 1 = 3
## [1] 3
```

八、 数据读写

我们经常会处理表格样式的数据，我们可以使用 read.table()函数进行读取数据。同样的如果我们需要保存我们处理好的表格时可用 write.table()函数进行保存。这部分相关内容将在课上进行详细的介绍。