# Ch2.7R编程

本章概要

*♢* **循环与向量化**

*♢* **用R写程序**

*♢* **编写自己的函数**

*♢* **养成良好的编程习惯**

**2.7.1循环和向量化**

**1．控制结构**

**1)条件语句**

**两种形式：**

* **if (条件) 表达式1 else 表达式2**
* **函数ifelse(条件，表达式1，表达式2)**

**例如：**

x<-c(2,0,-2)

if (x>=0) sqrt(x) else NA

ifelse(x>=0,sqrt(x),NA)

**2)循环结构**

**三种形式：**

* **使用for : for(变量in 向量) 表达式**
* **使用while : while (条件) 表达式**
* **使用repeat : repeat 表达式**

**区别：知道循环次数的用for循环；否则用while或repeat循环。**

**注：repeat循环中利用break语句跳出循环。表达式如果是一组命令（多个语句）时，要放在大括号里面。**

**例1：输出下列结果：**

**1**

**1 2**

**1 2 3**

**1 2 3 4**

**1 2 3 4 5**

**三种循环程序：**

for (i in 1:5) print (1:i)

i=1

while(i<=5) {

print (1:i)

i=i+1

}

i=1

repeat{

print(1:i)

i=i+1

if (i>5) break

}

例2：假定已有一个向量x，现在要产生一个长度与x相的同向量y，对于向量x中值为b的元素，y的对应赋0，否则赋1.

x<-c(12,43,5,67,90)

y<-numeric(length(x)) #创建一个与x等长度的数值型向量y

b=5

for (i in 1:length(x)) {

if (x[i]==b) y[i]<-0 else y[i]<-1

}

y

**2.向量化**

很多情况下可以利用向量化来避免循环结构和控制语句。向量化使得循环隐含在表达式中，条件语句可以用逻辑索引向量代替。如上面例2程序可以改写为：

x<-c(12,43,5,67,90)

y<-numeric(length(x)) #创建一个与x等长度的数值型向量y

b=5

y[x==b]<-0

y[x!=b]<-1

y

**向量化的好处:**

* 代码更简洁
* 提高计算的效率（因为C是一种编译语言，R是解释性语言，在计算时，通常C要比R快100倍。在R中使用向量化，R会立即调用C进行运算

**2.7.2用R写程序**

#例：对三种不同的鸟绘制类似的图，而且数据在三个不同的文件中

#方式一：一般做法

setwd("E:/Rprogram/data")

data<-read.table("Swal.dat")

plot(data$V1,data$V2,type="l")

title("Swallow")

data<-read.table("Wren.dat")

plot(data$V1,data$V2,type="l")

title("Wren")

data<-read.table("Dunn.dat")

plot(data$V1,data$V2,type="l")

title("dunnock")

#方式二：使用向量简化上述程序

layout(matrix(1:3,3,1))

setwd("E:/Rprogram/data")

species<-c("Swallow","Wren","Dunnock")

file<-c("Swal.dat","Wren.dat","Dunn.dat")

for (i in 1:length(species)) {

data<-read.table(file[i])

plot(data$V1,data$V2,type="l")

title(species[i])

}

注：如果程序保存在文件Mybirds.R中，可以通过函数source()来运行。

>source("E:/Rprogram/examples/Mybirds.R")

**2.7.3编写自己的函数**

编写自己定义的函数是学习R的主要目的之一，R软件提供的绝大多数函数，如mean(),var()等，是系统编写人员写的，与我们自己定义的函数本质上没有多大区别。

自定义函数的基本形式为：**name< - function(arg1,agr2,…) expression**

其中name为函数名，arg1,agr2,…是参数列表,expression是函数体,如果是多个语句需用{ }括起来。

函数调用格式：name(exp1,exp2,…)

注：调用自己编写的函数，需要将已写好的函数调用内存中。

**方式三：使用自定义函数**

myfun<-function(S,F){

data<-read.table(F)

plot(data$V1,data$V2,type="l")

title(S)

}

layout(matrix(1:3,3,1))

setwd("E:/Rprogram/data")

myfun("Swallow","Swal.dat")

myfun("Wren","Wren.dat")

myfun("Dunnock","Dunn.dat")

**方式四：向量化并使用mapply( )函数**

layout(matrix(1:3,3,1))

setwd("E:/Rprogram/data")

species<-c("Swallow","Wren","Dunnock")

file<-c("Swal.dat","Wren.dat","Dunn.dat")

mapply(species,myfun,file) #原书上程序有误, sapply(myfun,species,file)

**补充函数：apply(),lapply(),sapply(),mapply()**

**共同点：把函数应用到列表或数据框的每一个元素，或应用到矩阵的每一行或列上**

**不同点：前三者除函数名参数外，只能再传入一个参数，而mapply()可传入多个参数，而且参数位置不同**

x1<-c(1,2,3,4)

x2<-c(11,22,33,44)

M<-matrix(c(x1,x2),4,2)

D<-data.frame(x1,x2)

L<-list(x1,x2)

#求各列的和

apply(M,MARGIN=2,FUN=sum) #尝试：将M换成D、L能行吗？

lapply(L,FUN=sum) #尝试：将M换成D、L能行吗？

sapply(D,FUN=sum) #尝试：将M换成D、L能行吗？

mapply(FUN=sum,L) #尝试：将M换成D、L能行吗？

注1：函数的调用方式：

1）按位置调用 2）按名字（标签参数）调用 3）混合调用

以下三者等价：

apply(M,2,sum) #位置不可变

apply(X=M,MARGIN=2,FUN=sum) #位置可变

apply(M,FUN=sum,2)

注2：函数调用时的特性：采用定义时的默认值（见下面Ricker模型的例子)

例：Ricker模型研究种群动态变化相对于时间之间的关系图

Ricker 模型的定义如下:

这个模型广泛地用于种群动态变化的研究, 特别是鱼类的种群变化. 我们想用一个函数去模拟这个模型关于增长率*r* 和初始群体大小*N* 0相对时间的图表示. 我们还将设定一个可选项允许用户只显示最后若干步中种群大小(默认所有结果都会被绘制出来). 下面的函数就是Ricker 模型的数值模拟.

#先定义函数ricker()

ricker<-function(nzero,r,K=1,time=100,from=0,to=time){

cat("nzero=",nzero,"r=",r,"K=",K,"time=",time,"from=",from,"to=",to)

N<-numeric(time+1)

N[1]<-nzero

for (i in 1:time) N[i+1]<-N[i]\*exp(r\*(1-N[i]/K))

Time<-0:time

plot(Time,N,type="l",xlim=c(from,to))

}

#尝试函数的调用法：

ricker(0.1,1,2,50,15,120) #按位置调用函数

ricker(nzero=0.1,r=1,K=2,time=50,to=120,from=15) #按名字调用，注意to,from没按次序

ricker(0.1,1,2,50,15) #采用定义时的默认值

ricker(0.1) #没有默认值，参数赋值又不够时会出错

ricker(1,nzero=0.1,3,time=50) #名字、位置、默认定义混合用法

#再调用函数ricker()

layout(matrix(1:3,3,1))

ricker(0.1,1);title("r=1")

ricker(0.1,2);title("r=2")

ricker(0.1,3);title("r=3")

#补充1：自定义函数如果要返回值，是通过函数体中最后一个语句来实现的，返回值通常是向量、数组、矩阵、列表或数据框。

例如：对向量的各元素求和的自定义函数：

mysum<-function(a){

n<-length(a)

s<-0

for (i in 1:n){

s=s+a[i]

}

c(s) #和作为一个向量返回 ， 尝试：作为一个列表返回向量的长度与总和，list(Length=n,Sum=s)

#或者return(s)

}

mysum(c(1,2,3))

#补充2：自定义函数的参数变量是形式参数，且是局部变量，函数体中定义的变量也是局部变量；函数体外面的变量在函数体中是可用，即全程变量。因此里面的变量与外面变量同名时要小心

mysum<-function(a){

print(a) #参数中的变量是形式参数

print(s) #外面的变量在函数体中可用

n<-length(a)

s<-0 #一定要清零

for (i in 1:n){

s=s+a[i]

}

return(list(Length=n,Sum=s)) #返回列表

}

a=c(0,0,0)

s=10 #函数里面的变量与外面变量同名时要格外小心，容易出错

mysum(c(1,2,3))

print(a)

print(s)

print(n) #n是局部有效

**2.7.4养成良好的编程习惯**

**好程序的标准：**

* 可读性
* 可理解性

**四个良好的编程习惯：**

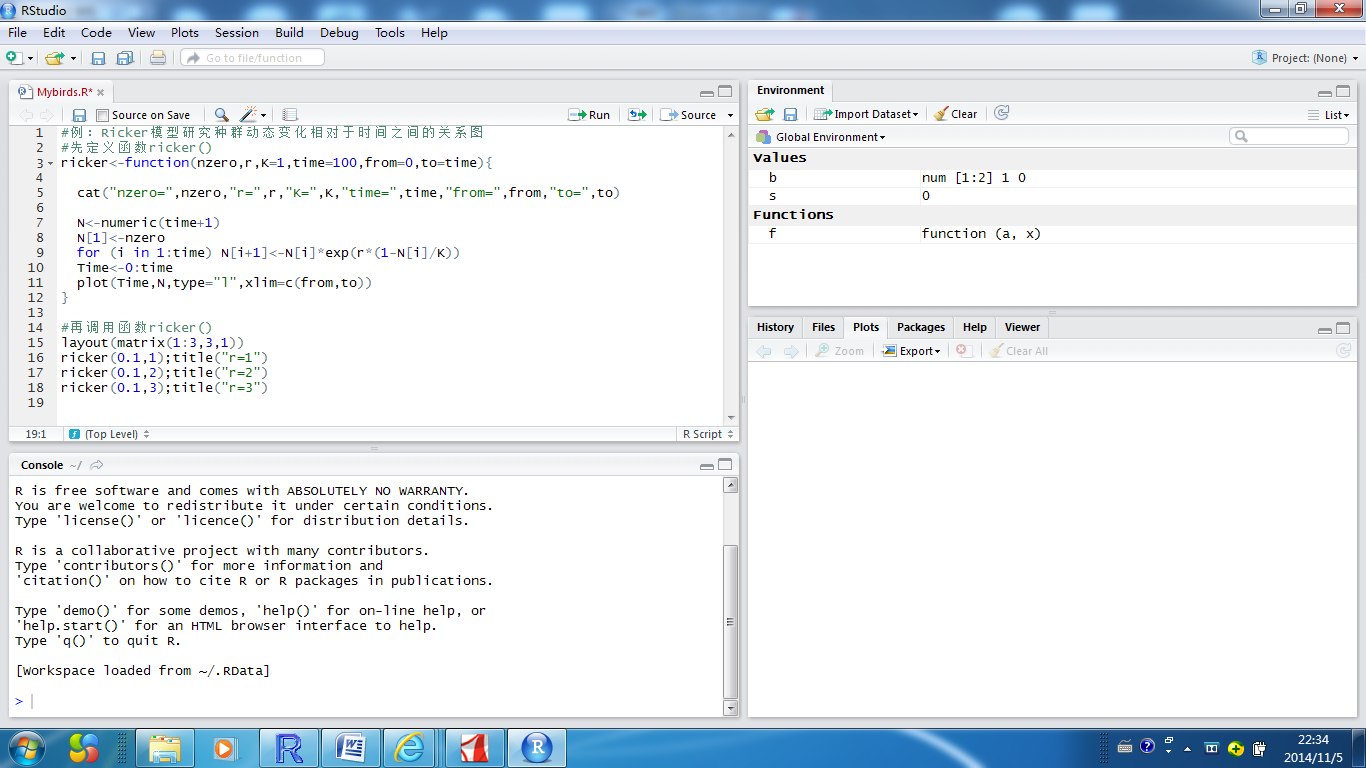
习惯之一**:** 采用结构化、模块化编程;

习惯之二**:** 增加注释(Commenting), **R** 中使用# 作为注释语句的开始;

习惯之三**:** 使用意义明确的名字给变量命名, 切忌使用人或宠物的名字;

习惯之四: 行前自动缩进(Indentation)，推荐R语言的集成开发软件Rstudio。

官网：http://www.rstudio.com



本节学习的R函数小结

|  |  |  |
| --- | --- | --- |
| 第2章 2.7 R的编程所介绍的主要函数 | | |
| 函数 | 功能 | 示例 |
|  |  |  |
|  |  |  |

**作业：练习本节所有命令**

* **2.5**
* **2.6**