



数据分析与处理技术——函数式编程

商学院 徐宁

函数式编程

1.流程控制

循环流程

分支流程

混合嵌套

循环与遍历

for函数使得其后{ }中的代码的反复执行，如下循环中，**i**称为循环标志变量，用来控制循环次数。

```
y=0  
for(i in 1:100){  
    y=y+i  
}
```

括号{ }用于圈定被作用的代码段

循环流程是{}中代码重复执行

其他循环方式

repeat无限重复式循环

```
y=0
i=1
repeat{
    y=y+i
    i=i+1
    if(i>100) break
}
```

repeat循环需自行
设置代码段的出口

while条件判断式循环

```
y=0
i=1
while(i<=100){
    y=y+i
    i=i+1
}
```

while循环需自行调
节循环标志变量

任何循环均不得出现死循环，否则将无法执行

循环与遍历

遍历数据集是循环的重要用途之一，如下例子，对**persons**数据集中每个对象进行处理，循环标志变量**i**被用于行索引

```
n=nrow(persons)
for(i in 1:n){
  persons[i,"Math"]=persons[i,"Math"]+1
}
```

循环流程不同于向量化运算，循环会逐个处理对象，通过**for**的控制实现代码的持续重复执行。

分支流程

分支流程通过**if**引导的条件进行判断，对待选择的{ }中代码块选择性执行或不执行。

```
if(t>50){  
    cat('t is bigger than 50')  
}else  
{  
    cat('t is smaller than 50')  
}
```

分支流程的结构：

```
if(条件) {  
    statement1  
}  
else{  
    statement2  
}
```

其中**else**可省略，也可换作**elseif**进一步嵌套条件

混合嵌套流程

循环与分支通过混合嵌套能够解决大量现实中复杂繁琐的计算问题

```
n=nrow(persons)
for(i in 1:n){
  if(persons[i,"Math"]<60){
    persons[i,"Math"]=persons[i,"Math"]+5
  }else{
    persons[i,"Math"]=persons[i,"Math"]+1
  }
}
```

$$\sum_{j=1}^{10} \sum_{i=1}^{10} i \sin(j \cdot \pi)$$

1000以内能被3整除的所有正整数相加等于多少？

函数式编程

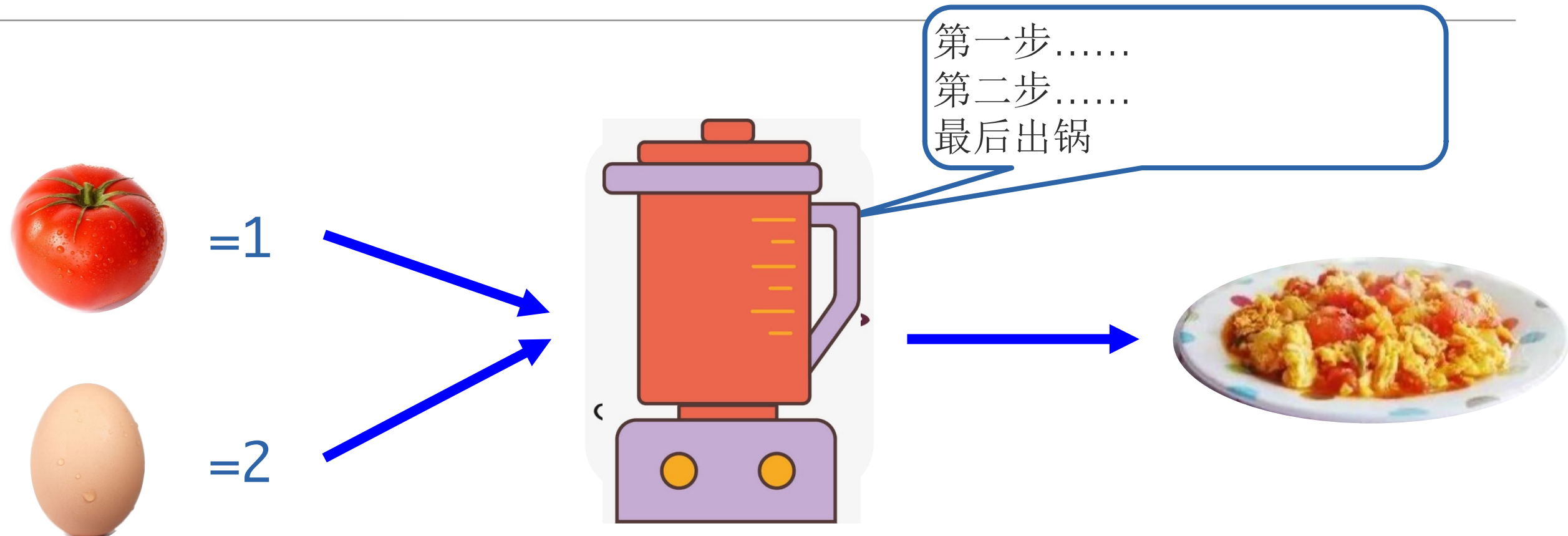
函数创建格式

参数和返回值

2. 自定义函数

自动化处理重复出现的问题

10



重复出现的问题：投入原料是什么；处理过程是什么；得到的结果是什么。

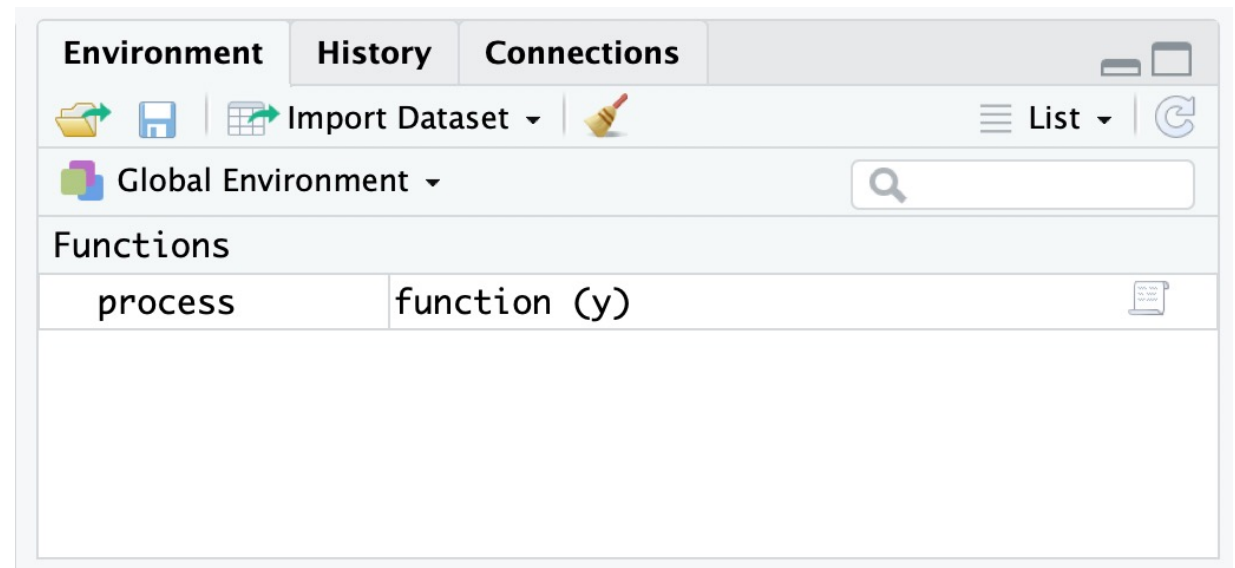
函数的创建

函数将计算过程进行封装，减轻重复的计算工作。

`process()`

```
t=y[y>60]  
dist=mean(t)-median(t)  
return(dist)
```

```
> y=c(10,12,70,80,95,23)  
> process(y)  
[1] 1.666667
```



自定义函数被存放在全局环境中

函数的创建

使用function()创建函数

```
函数名=function(参数)
{
    ....
    return(计算结果)
}
```

```
> process <- function(y){
+   t = y[y > 60]
+   dist = mean(t) - median(t)
+   return(dist)
+ }
```

```
> w = c(50, 70, 80, 90)
> x = process(y = w)
> x
[1] 1.666667
```

函数返回值

函数可以不返回结果

```
> process <- function(y){  
+   t = y[y > 60]  
+   dist = mean(t) - median(t)  
+   print(dist)  
+ }
```

```
> process(y)  
[1] 1.666667
```

函数返回值

返回值可以是任意数据类型

```
> process <- function(y){  
+   t = y[y > 60]  
+   dist = max(t) - min(t)  
+   var = list(y1 = dist, y2 = median(t))  
+   return(var)  
+}  
  
> x = process(y)  
> x  
$y1  
[1] 20  
$y2  
[1] 80  
> class(x)  
[1] "list"
```

参数传递

函数内部是封闭的

参数传递数据到函数内部

函数允许设置缺省参数

```
> process <- function(y, x=60){  
+   t=y[y>x]  
+   dist=mean(t)-median(t)  
+   return(dist)  
+}
```

```
> process(y=2)  
[1] 4
```

```
> process()  
[1] 9
```

练习：编写计算函数

采购部经理找到数据分析员，要对供应商评估选择过程进行自动化处理。每个供应商有五项打分，各项满分均为**100**，及格分为**60**。计算方法：

1. 去掉所有低于**60**分的分数；
2. 在剩下的分数中计算 最终得分=平均值*最低分

解决方式

处理步骤

第一步：筛选保留大于60分的成绩；

第二部：计算平均值与最小值的乘积

对应的R语言代码

```
process=function(y){  
  t=y[y>60]  
  score=mean(t)*min(t)  
  return(score)  
}
```

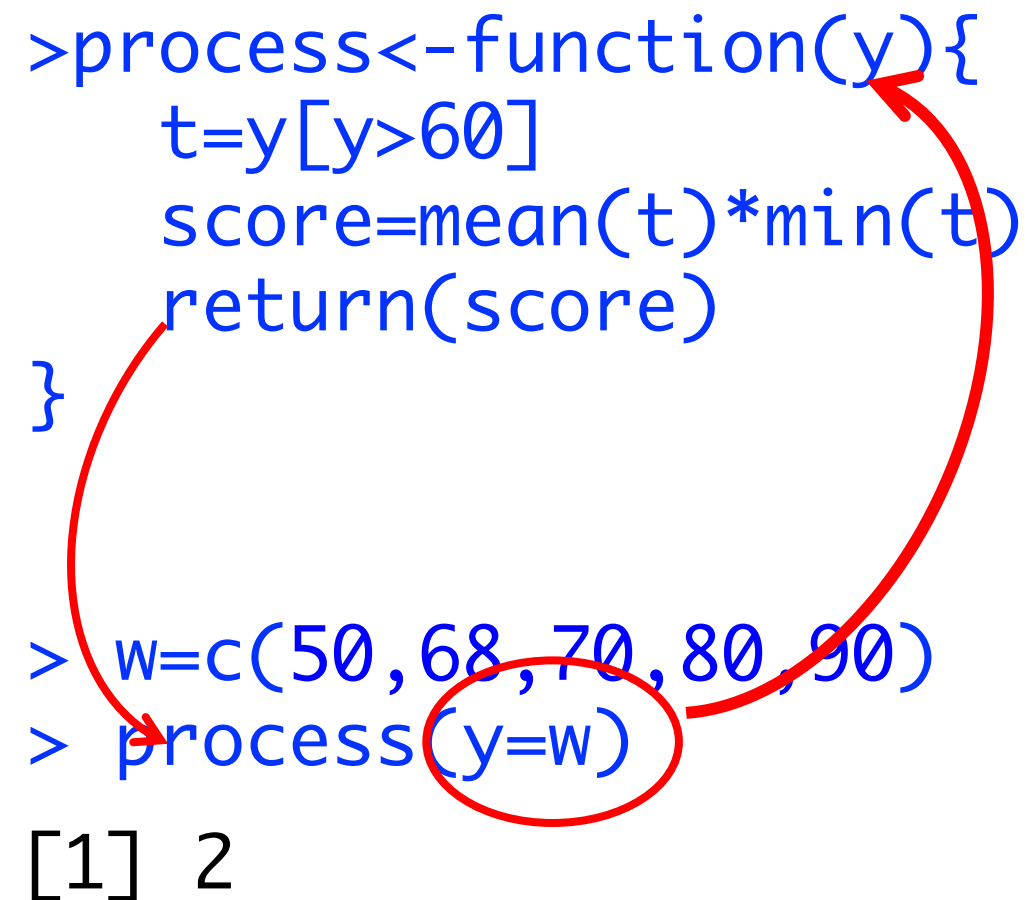
函数使用方法

要点：函数的创建方法

```
函数名=function(参数)
{
    ....
    return(计算结果)
}
```

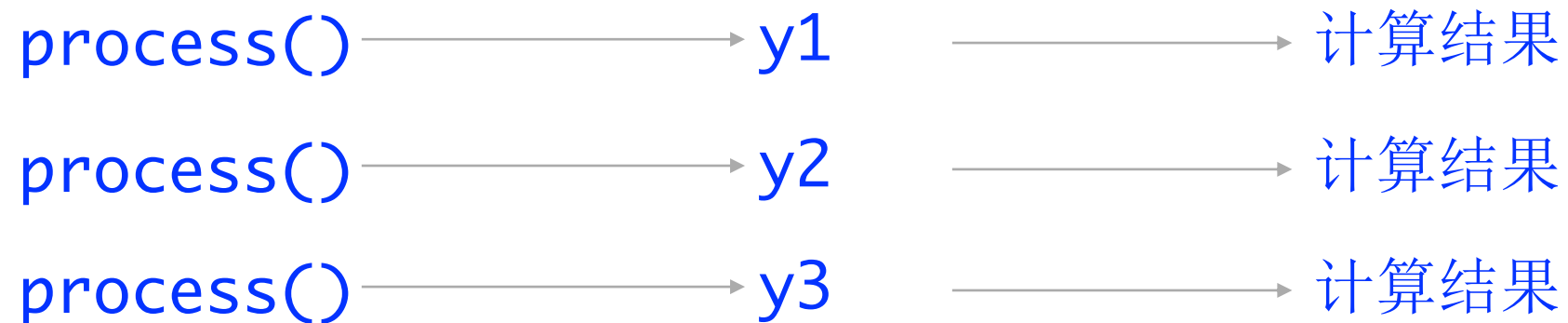
```
> process <- function(y) {
  t = y[y > 60]
  score = mean(t) * min(t)
  return(score)
}

> w = c(50, 68, 70, 80, 90)
> process(y = w)
[1] 2
```



函数式处理数据的过程

处理三个供应商的打分过程示意图



总结：函数整合了一套固定的计算流程，仅利用括号（）作为传递数据的接口。

函数式编程

3.环境的作用域

环境结构

环境作用方式

函数环境

环境结构

环境的特征：

环境呈链式结构

不同环境中的变量互不干扰

全局环境处于环境链最底层


环境中的变量：

变量不能重名，不可以用数字或特殊符号开头；

函数与变量可以重名

函数与变量优先于工具包中的同名对象

全局环境会搜索缺失函数或变量



The screenshot shows the R Environment window with the 'Global Environment' selected. It displays a table of variables and their values. A red circle highlights the table content.

Data	
e1	Environment
Values	
a	7L
b	6.2
e	0.86155
x	chr [1:3] "apple pie" "apple" "apple ...

环境是存放变量的地方

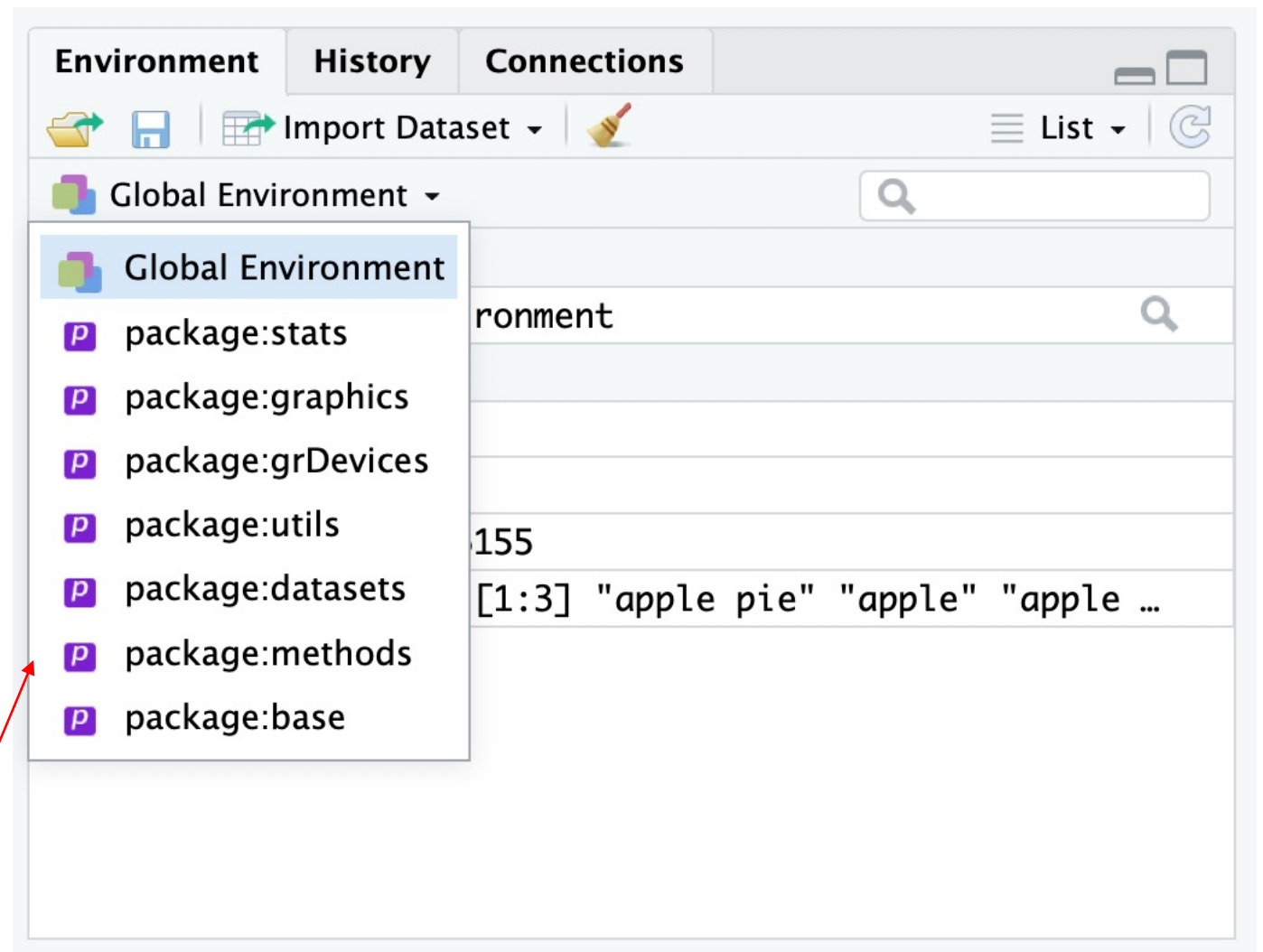
环境结构

环境的特征：

环境呈链式结构

不同环境中的变量互不干扰

全局环境处于最底层



环境是分层的，这样可以方便工具包、变量、函数错落有致的一起工作。

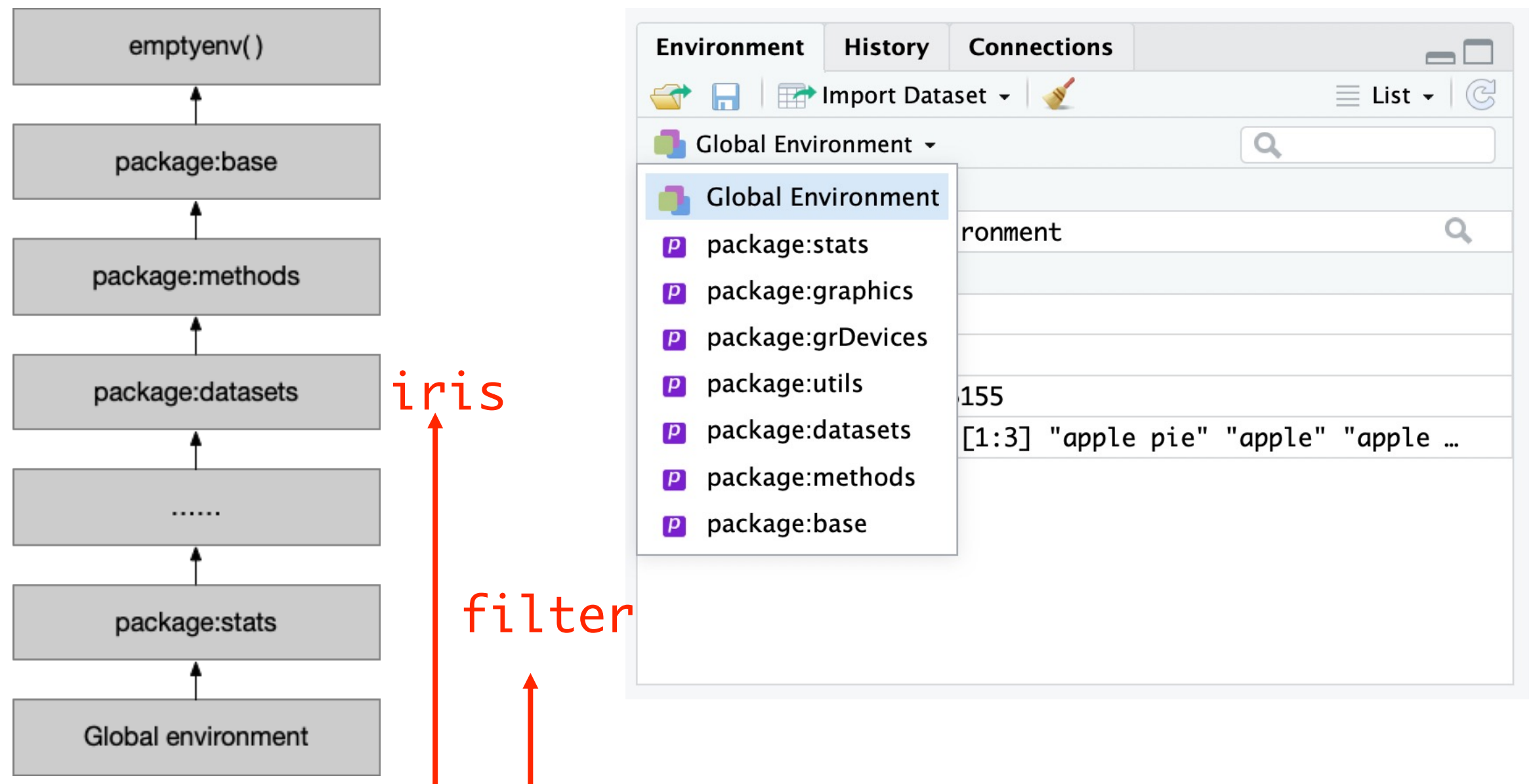
环境操作

- `ls()` 列出环境中的对象
- `rm()` 删除环境中某个对象
- `search()` 搜索全局环境的所有父环境

```
> a=1:10
> rm(a)
> rm(list=ls())
> search()
[1] ".GlobalEnv"
"tools:rstudio"
[3] "package:stats"
"package:graphics"
[5] "package:grDevices"
"package:utils"
[7] "package:datasets"
"package:methods"
[9] "Autoloads"
"package:base"
>
```

环境的逻辑链

调用iris数据集、filter函数时沿环境逻辑链搜索过程



工具包环境

动态加载的工具包处于紧邻全局环境之上

```
> library(dplyr)
```

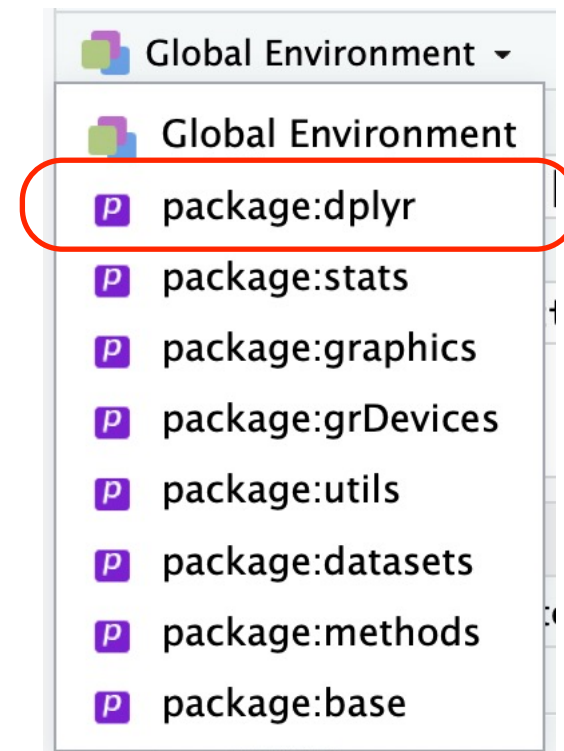
Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

`filter`, `lag`

指定工具包环境调取函数

```
> stats::filter()
```

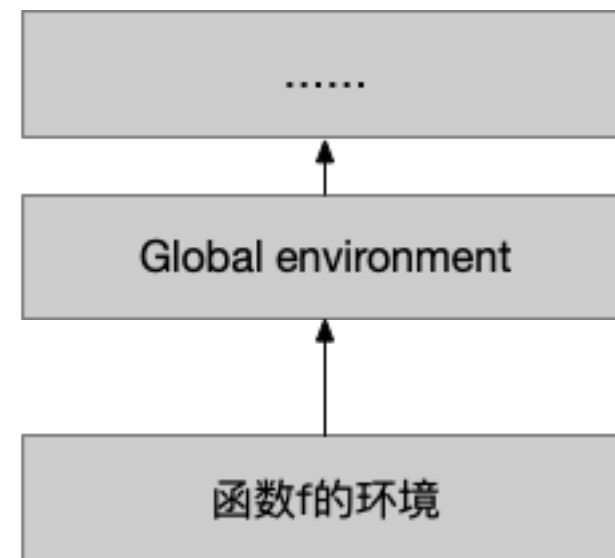


函数的环境

函数环境的特点：

函数的环境仅在计算时存在
函数环境处于全局环境之下
函数的环境仅在计算时存在

```
f=function(y){  
  dist=max(t)-min(t)  
  return(dist)  
}
```



变量作用域

变量作用域规则:

1. 向上查找
2. 动态查找
3. 屏蔽效应

`<<-` 在函数内操作位于全局的变量

```
> x=12
> g=function(){
+ z=7
+ c(x,z)
+ }
> z=8
> g()
[1] 12  7
```

```
g <- function(){
  z=7
  a<<-15  #对全局环境中复制变量a
  return(c(x,z))
}
```

函数式编程

4. 泛函

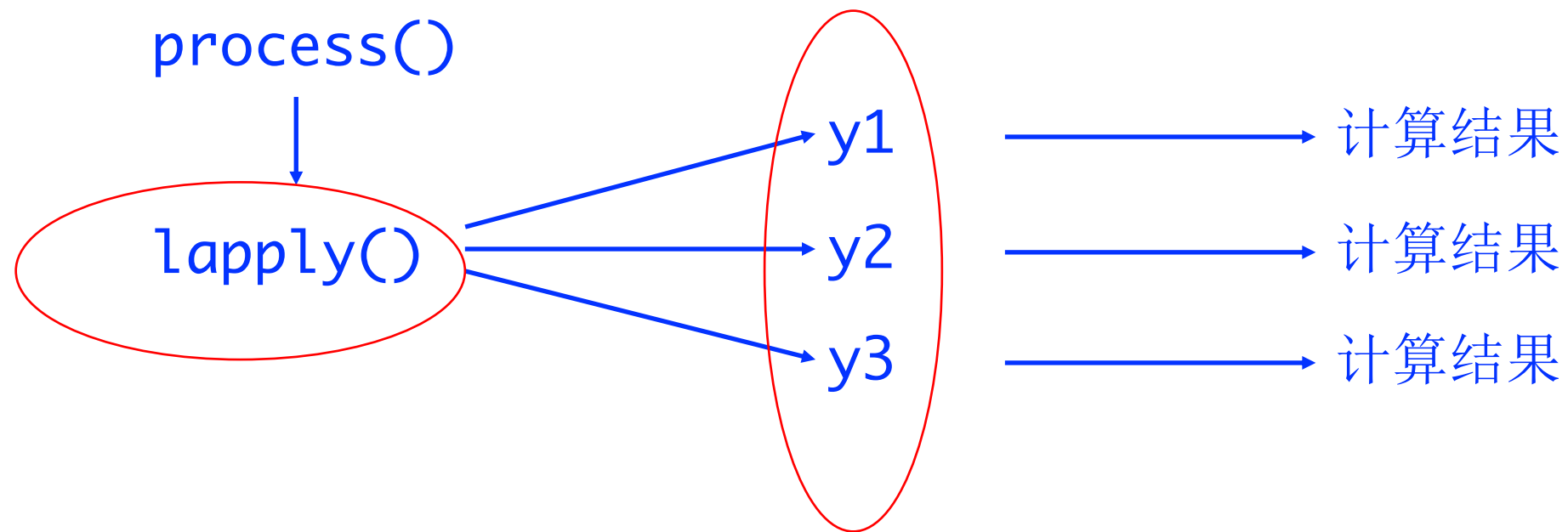
lapply

切割与组合

泛函族

使用函数之函数处理复杂问题

重新调整函数使用过程



泛函方法：将函数作为参数施加到目标数据列表之上的方法。

注意：`lapply`是`apply`系列泛函族中最常用一个，其他泛函原理相同，请自行查阅。

泛函lapply

`lapply(data, f, ...)`

参数解释：

data 列表格式目标数据集
f 调用的函数
... 被调用函数的参数

泛函的优势在于可以利用处理器多核并行运算，成倍提高计算速度。

已导入数据到R语言工作环境的lt变量中

Data	
lt	List of 3
: num [1:4]	50 70 80 90
: num [1:5]	40 60 90 85 70
: num [1:6]	12 77 84 32 90 95

```
▶ lapply(lt, process)
```

```
[[1]]
[1] 20
```

```
[[2]]
[1] 20
```

```
[[3]]
[1] 18
```

plyr泛函族

- Split-apply-combination
- **plyr**将切割-使用-组合做了完美整合，注意下表中**plyr**泛函名称变化

输入格式

输出格式

	array	data.frame	list
array	aapply	dapply	lapply
data.frame	adply	ddply	ldply
list	alply	dlply	llply
_(无输出)	a_ply	d_ply	l_ply

函数式编程

5.特殊函数形式

中缀函数

替换函数

函数列表

闭包

中缀函数

“一切皆为函数”，包括 $+$ $-$ $*$ $/$ 等运算符。中缀函数，即运算符，以函数名两端变量为第一、第二参数。

案例：创建一个名为 `%^_^%` 的运算符，实现如下运算

`2%^_^%3` 即 $2*3+2^3$

```
"%^_^%"=function(a,b){  
    s=a*b+a^b  
    return(s)  
}
```

函数即为运算符形式：

`2%^_^%3`

本质上，中缀函数将函数流程绑定在了字符 `%^_^%` 之上。

替换函数

替换函数的方式似乎是在对函数作用结果进行替代赋值。如

`names()`函数:

创建一个向量`t`

`t=1:10`

用`names()`更改元素名:

`names(t)=letters[1:10]`

案例: 创建一个名为`second`的替代函数

```
"second<-"=function(x,value){  
  x[2]=value  
  return(x)  
}
```

运行函数之后起到替换第二个元素的作用。

`second(t)=50`

本质上, 替换函数将函数流程绑定在了带赋值符的字符上。

函数列表

函数也是可以作为元素存入变量。`list` 变量可以将函数像数据一样存储。

如前所述`process`函数，存入变量，同时把`mean`函数也存储

```
>process<-function(y){  
+   t=y[y>60]  
+   dist=mean(t)-median(t)  
+   return(dist)  
+}
```

```
func=list(a=process,b=mean)
```

`func`称为函数列表，存入其中的函数使用时如同调取数据一样：

```
func$a(y=c(50,70,80,90))
```

```
func$b(1:5)
```

闭包

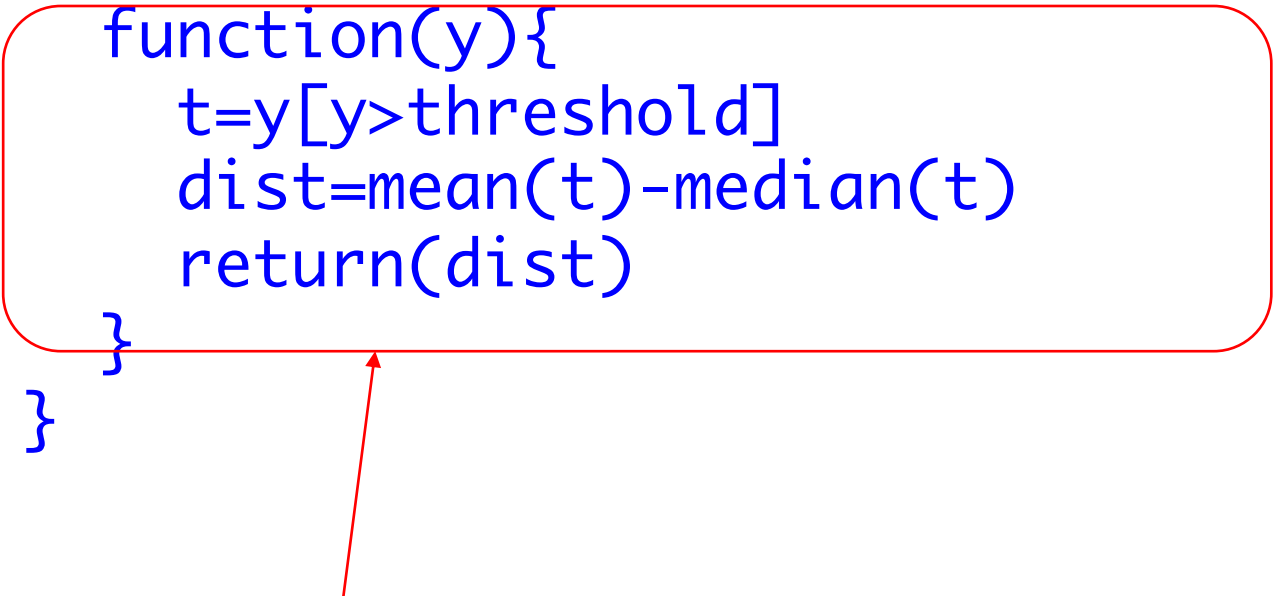
闭包：函数的返回值也是函数，即生产函数的函数。

以`process`函数为例，

```
>process<-function(y){  
+   t=y[y>60]  
+   dist=mean(t)-median(t)  
+   return(dist)  
+}
```

创建一个闭包，用来生产一批函数`process`，区别就在于其中的筛选标准不再是`60`，而是给定参数`threshold`，如下闭包：

```
factory=function(threshold){  
  function(y){  
    t=y[y>threshold]  
    dist=mean(t)-median(t)  
    return(dist)  
  }  
}
```



这个结构被作为返回值赋予了结果变量`factory`。

练习1：编写函数

- 编写一个函数`process`，要求任意输入一个日期数据，计算出距离今天经过了几周，其中不足一周的按一周算。
- 创建一个函数`process`，若输入变量是数值型矩阵，则找出最大元素和最小元素，以原子向量形式返回结果，若不是，则返回`0`。

练习2：函数的参数与环境

全局环境中存在 $a=10, b=12, c=17$, 编写函数 g , 并执行计算 $g(3)$, 分析函数结果。

```
g=function(a,c=3){  
    b=5  
    s=c(a,b,c)  
    return(s)  
}
```

练习3：横加运算函数

创建函数 **f**，实现横加运算，即将输入参数的整数位上各数字相加的功能，例如对于数字 **215**，**myfunc(215)** 结果为 **8**，即 **2+1+5**。若输入参数为负则取绝对值，若为非整数则取整。计算向量 **a=c(10,-15.218,77)** 的横加结果。

练习4：计算学生成绩的绩点

persons数据集记录了学生基本信息（年级Grade, 年龄Age, 数学成绩Math, 英语成绩English, 计算机成绩Computer）

其中英语分为5档打分，ABCDE折算百分制分别为95，85，75，65，55。学生绩点计算方法为：

- 1年级绩点计算公式为： $0.4 * \text{数学} + 0.3 * \text{英语} + 0.3 * \text{计算机}$
- 2年级绩点计算公式为： $0.3 * \text{数学} + 0.2 * \text{英语} + 0.5 * \text{计算机}$
- 若单科成绩不及格（低于60分）则该科目在绩点中记0分。

请编写程序完成对全部学生绩点的计算并给出排序。

练习5：合并运力矩阵

excel文件huowu.xlsx记录了2017年某饮料公司重要物流结点运量数据，其中行名称代表出发地，列名称代表目的地。现要建立7大区域中心，请将城际运量整合为区域运量。

合并成7大区域：

东北：沈阳，长春，哈尔滨

西北：西安，银川，兰州，西宁，乌鲁木齐

华北：北京，天津，石家庄，太原，济南，青岛，呼和浩特

华东：上海，南京，杭州，合肥

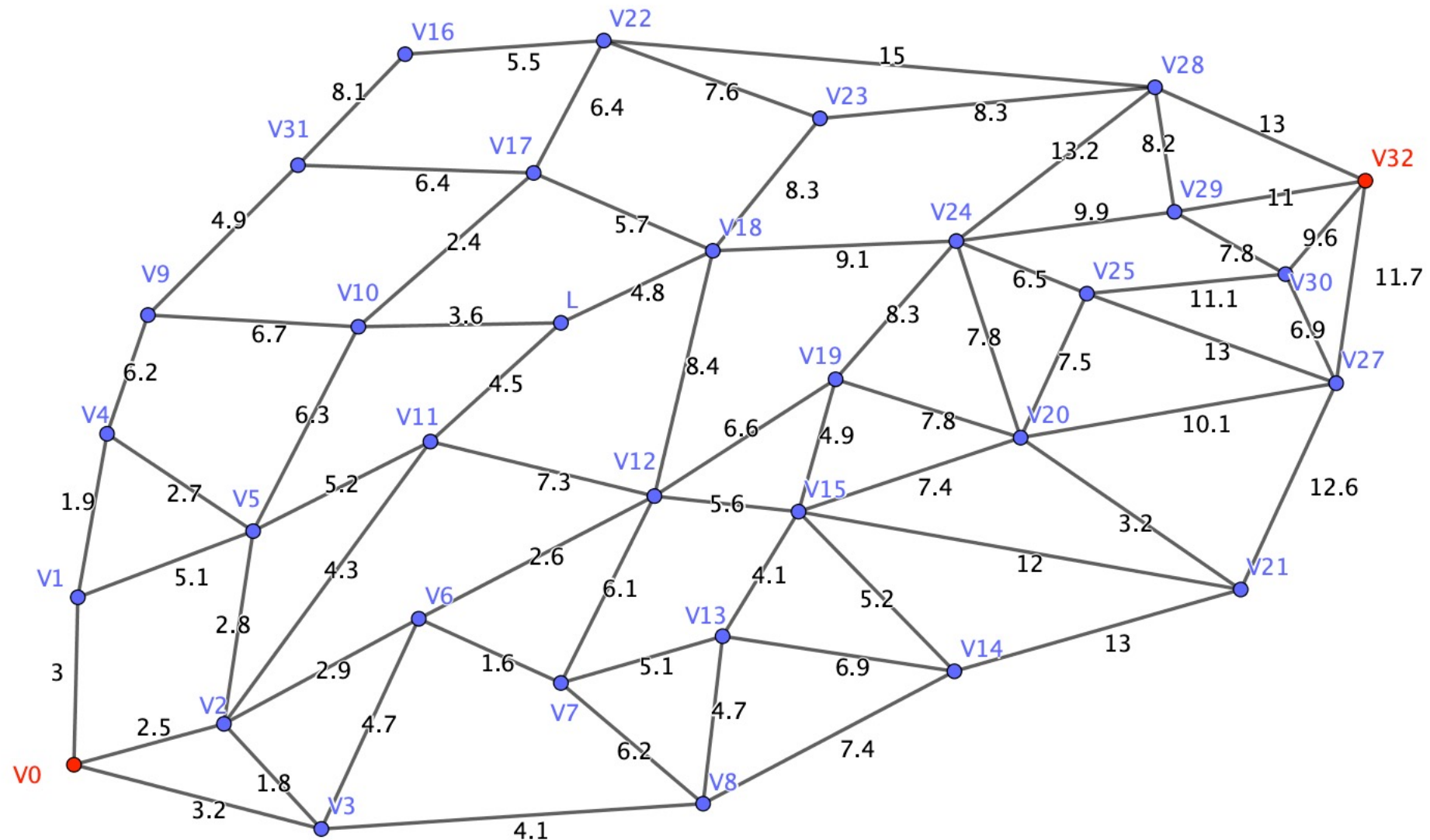
华中：武汉，郑州，长沙，南昌

华南：广州，深圳，南宁，海口，福州，厦门

西南：成都，重庆，贵阳，昆明，拉萨

练习6：动态规划寻找最短路径

编写程序，找出从V0出发到终点V32的最短路径



练习7：粒子群算法

粒子群算法(简称PSO)是群体智能算法的一种经典算法，模仿自然界中生物个体组成的种群表现出的群体智慧，用以解决寻找函数最优值的问题。该类算法以随机搜索方式逐步逼近函数最优值位置。请搜索以下函数在 $[-10, 10]$ 区域内最小值及其位置，说明算法的精确度。

$$f(x_1, x_2) = x_1^2 + x_2^2 - 10 \cos(2\pi x_1) - \cos(2\pi x_2) + 10$$

*粒子群算法参考流程

适应度函数为： $f(X)$ ，其中 $X = (x_1, x_2, \dots, x_D)$ ，上限设为 $U = (u_1, u_2, \dots, u_D)$ ，下限 $L = (l_1, l_2, \dots, l_D)$ 。算法设置种群数量为 NP ，即 $X_1, X_2, X_3 \dots X_{NP}$ ，其中第 i 个粒子为 $X_i = (x_{1i}, x_{2i}, \dots, x_{Di})$ ，粒子群算法步骤如下：

第一步：初始化，利用随机数随机生成 NP 个粒子，设置迭代搜索次数 M

第二步：计算所有粒子适应度函数 $f(X_i)$ ，并记录下本次搜索中最小适应度的粒子位置**Pbest**，对比历次搜索最小适应度位置**Gbest**，若 $f(Pbest) < f(Gbest)$ ，则令**Gbest**更换为**Pbest**。

第三步：改变粒子位置，计算粒子 i 的运动速度：

$$v_i = \alpha(Gbest - X_i) + \beta(Pbest - X_i) + \gamma R_i$$

其中 α β γ 为自行设定的参数，用于调节速率，通常设置范围为 $(0,1]$ ， R_i 为 D 维随机数。利用速度 v_i 对粒子位置更新：

$$X_i = X_i + v_i$$

第四步：所有粒子位置更新后返回第二步，开始新一轮搜索，并再次记录和比较**Pbest**和**Gbest**取值。若达到最大搜索次数 M ，则输出最优解的 X_{Gbest} 和极值 $f(X_{Gbest})$ 。