

数据分析与处理技术

函数作用机制

什么是函数

R是由各种扩展包package组成的，每一个包中的工具都以函数形式装载，即function。

函数就像一个工厂：原材料以参数形式传递给函数，函数在一个看不到的环境运行计算，完成后将生产成品作为结果交付给你(全局环境)。

例如，`y=sin(10)` `mean(1:10)` `plot(cars)`等

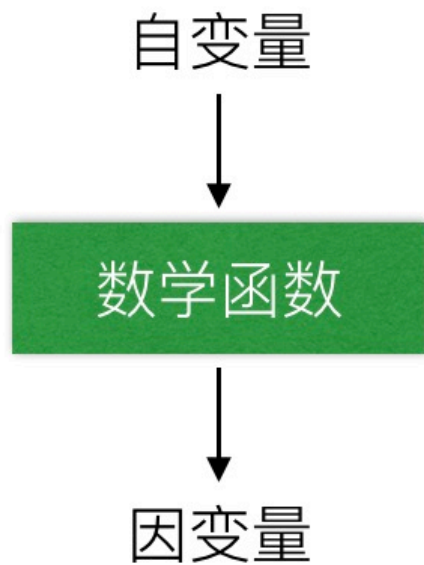
而事实上，R中所用到的一切命令都是函数，甚至包括`+` `-` `<-`等符号

直接输入函数名并且不带 `()` ,将调取函数的代码，即看到函数内部构成

```
> sd
function (x, na.rm = FALSE)
sqrt(var(if (is.vector(x) || is.factor(x)) x else as.double(x),
          na.rm = na.rm))
<bytecode: 0x107fee550>
<environment: namespace:stats>
```

注意：某些基础包中的函数并非由R写成

函数使用



函数使用的基本方法：

变量 <- 函数名(参数名=参数值, 参数名=参数值, ...)

```
> plot(x=cars$speed,y=cars$dist)
```

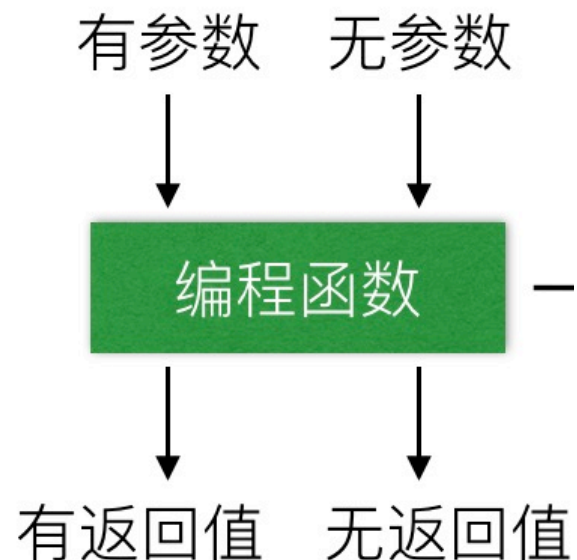
Usage

```
> plot(cars)
```

`plot(x, y, ...)`

按照顺序输入参数可以省略参数名，参数值将按顺序赋予参数

思考：函数data.frame()如果省略参数名会有什么不同？



编写函数

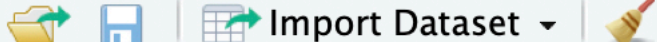

函数基本结构： 函数名 \leftarrow function(参数1, 参数2)

```
{  
  .....  
  .....  
  return(变量)  
}
```

```
> myfunc <- function(a,b){  
+ c=a^2+b%%2  
+ return(c)  
+ }  
> myfunc(a=2,b=7)  
[1] 5
```

自定义函数myfunc,将对参数a和b的值进行计算生成变量c, 最后把c作为结果返回

函数运行环境是隔离的, 仅通过参数进行数据传递, {}中间是函数的计算代码, 最后的return()将指定把那个变量作为输出结果返回到全局环境, 否则将以最后一个变量调用命令为返回结果

Environment	History	Connections
 Import Dataset ▾		
 Global Environment ▾		
Functions		
myfunc	function (a, b)	

此时, myfunc将作为全局环境中的函数存在, 而它的计算环境则是调用时生成 计算完成后丢掉。

参数传递

由于函数运行的环境是相对封闭的，函数内部代码使用的变量与全局环境不相干，两者交换数据的地方主要是参数。

```
> h<- function(a,b){  
+ c(a,b)  
+ }
```

编写函数时可以直接写入默认参数，当调用者没有填写参数时直接使用默认参数值

```
> f<-function(a=1,b=a*2){  
+ c(a,b)  
+ }  
.
```

...是泛型参数，即将无法识别的参数带入函数内部处理，而非以报错方式中止函数运行

```
> f_test<-function(...){  
+ names(list(...))  
+ }  
> f_test(a=1,b=3)  
[1] "a" "b"
```

由于提前并不知道...会接收多少参数，通常可以用`list()`这种易于使用的方式捕捉...中的参数信息。

向量化运算

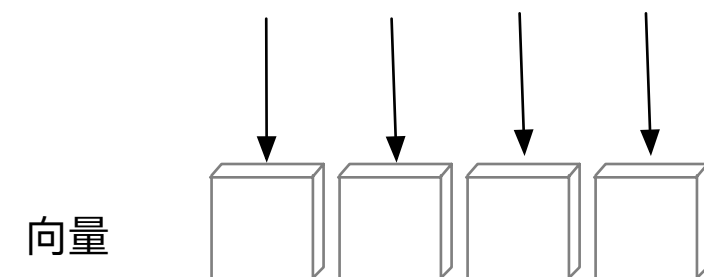
函数被比喻为来料加工的工厂，那么工厂总是更为喜欢批量化生产，对应函数中就是向量化运算。

由于R的原子向量结构支撑了向量化运算，即一次性运算所有向量中数据。只要函数内部没有迭代运算，自创函数也是支持向量化运算。这在复杂算法中非常重要，能够充分利用计算机资源快速批量处理数据计算。

```
> f(3)
[1] 5
```

```
> f
function(x) x+2
```

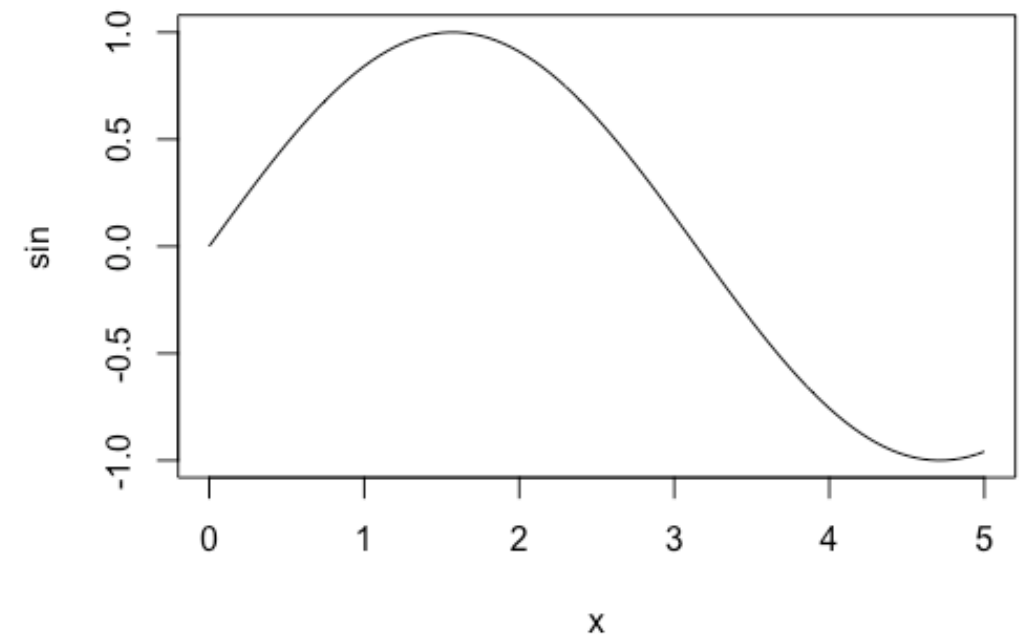
```
> f(c(1,3,5,7,9))
[1] 3 5 7 9 11
```



泛函使用

当函数将另一个函数作为参数调用时，这种函数称为泛函，我们已经接触过的泛函有dplyr包里的summarise，以及apply,lapply等。
包括plot也是可以当作泛函使用

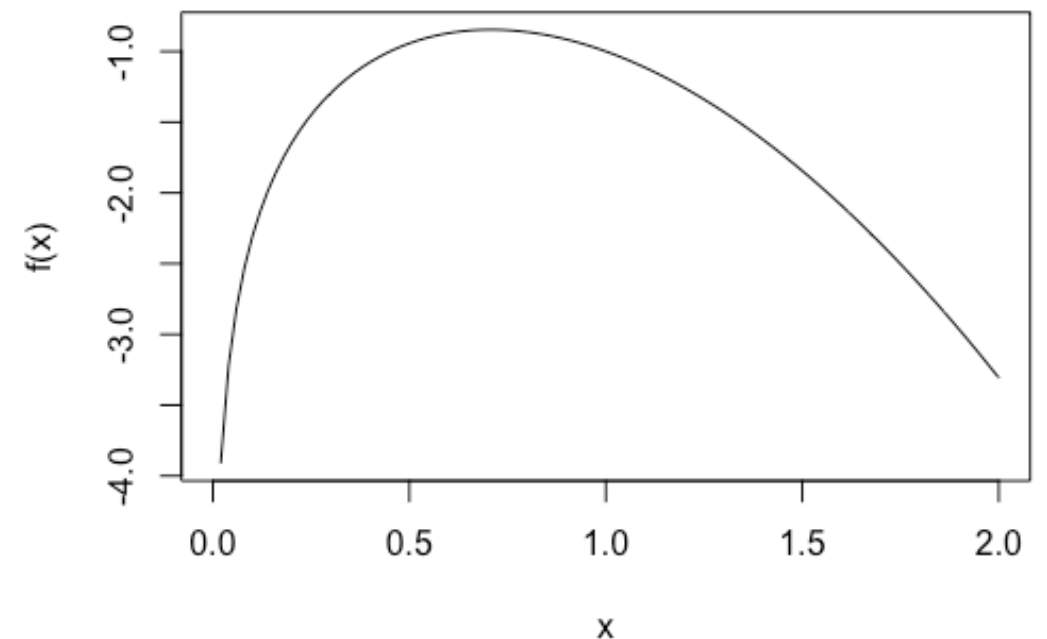
```
> plot(sin,0,5)
```



与值相同的是curve函数，即画曲线函数

```
> f=function(x) log(x)-x^2
```

```
> curve(f,0,2)
```



函数存储

函数可以被当作一个数据装入变量当中，能够装载函数的变量只有list类型。

```
> f<- function(x) x+2
> f(3)
[1] 5
> cc
[1] "北京"
> an
[1] 1 2 3 NA
> a<- list(cc, an, f)

> a[[3]]
function(x) x+2
```

函数f被装入的列表变量a中，而它的读取和使用则按照列表元素的方式使用。

函数结构

所有R的函数包括三个部分：

`body()`，函数内部代码

`formal()`，控制如何调用函数的列表

`environment()`，函数变量为止的“地图”

使用C语言编写的原函数被系统使

用`.Primitive()`调用，而非使用R代码，所以

`sum` `mean`这些函数的三部全部显示NULL

```
> myfunc <- function(a,b){  
+ c=a^2+b%%2  
+ return(c)  
+ }  
> myfunc(a=2,b=7)  
[1] 5
```

```
> body(myfunc)  
{  
  c = a^2 + b%%2  
  return(c)  
}  
> formals(myfunc)  
$a
```

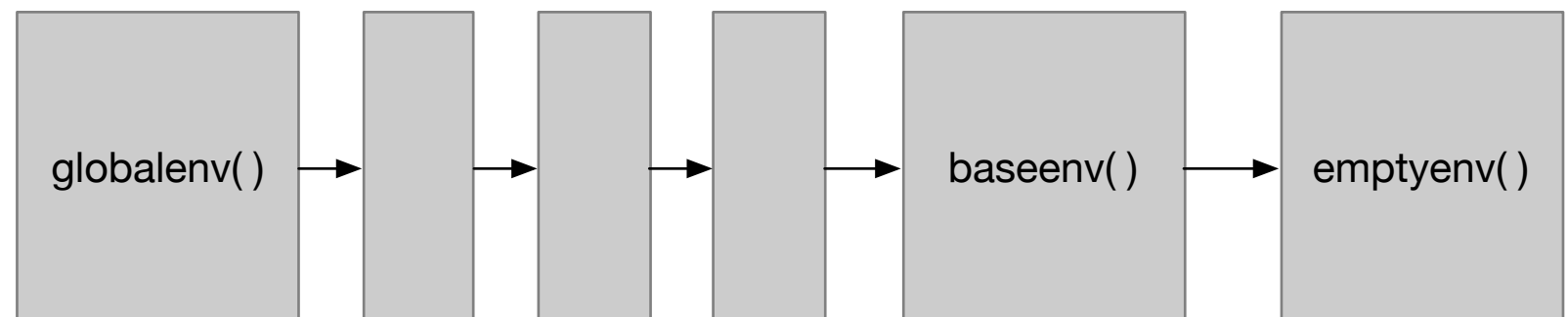
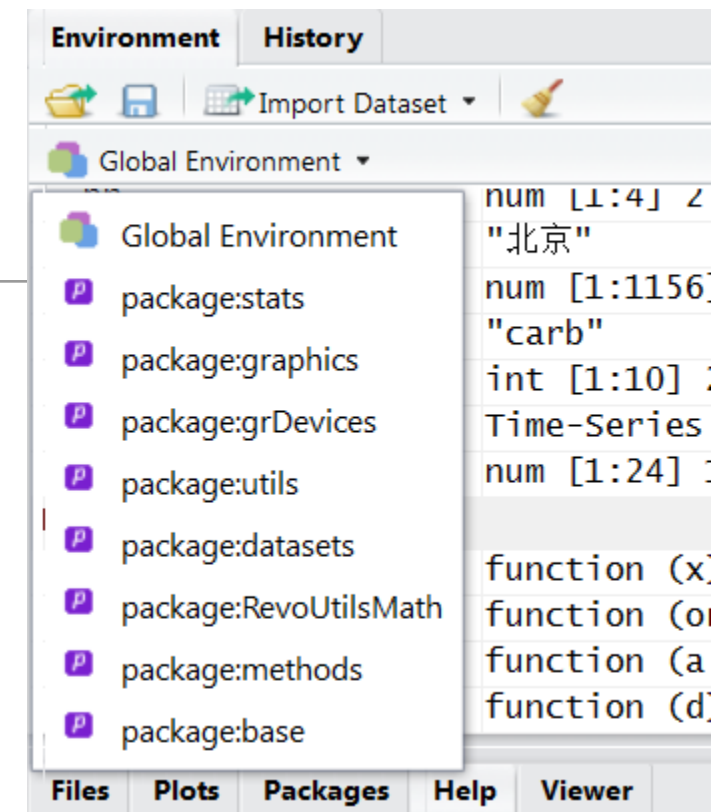
```
$b
```

```
> environment(myfunc)  
<environment: R_GlobalEnv>
```

环境

环境就像一个口袋, 里边装满了各种名字,
环境的作用就是将名字与数值进行绑定。

每一个扩展包都有它自己的环境, 创建在全局环境
的函数在运行时也有它自己的运行环境。



```
> globalenv()
<environment: R_GlobalEnv>
> env<-globalenv()
> class(env)
[1] "environment"
> search()
```

搜索路径

```
[1] ".GlobalEnv"      "tools:rstudio"    "package:stats"    "package:graphics"
[5] "package:grDevices" "package:utils"    "package:datasets" "package:methods"
[9] "Autoloads"       "package:base"
```

词法作用域

词法作用域是一组规则，R遵守这个规则去寻找需要的变量名称。

思考：创建函数f如下

```
> x<-12  
> g<-function(){  
+   z<- 7  
+   c(x,z)  
+ }  
.  
  
> z=8  
> g()
```

g的运行结果是多少？

词法作用域四个查找规则：

- 1.向上查找：先在函数内部查找，再去上一层环境查找
- 2.名字屏蔽：全局环境中的变量与函数环境中的不冲突
- 3.重新开始：每次调用都重新建立环境
- 4.动态查找：运行时查找而不是创建时查找（即词法作用域决定在哪里查找，却不决定何时查找）

根据以上规则再对照思考题中的g()应当取什么值。

中缀函数

中缀函数是函数的一种特殊形式，以加号为例

```
> 1+2  
[1] 3
```

加号+就是一种中缀函数，改为前缀调用方式：

```
> "+"(1,2)  
[1] 3
```

编写中缀函数相当于编写运算符，但函数参数仅有两个，例如：

```
> "%^_%"<-function(a,b){  
+ a1<-a*b + b1<-a^b  
+ a1+b1  
+ }  
  
> 2%^_%^3  
[1] 14
```

替换函数

替换函数是另一种函数特殊形式，以names()例

```
> t<-1:10  
> names(t)<-letters[1:10]  
> t
```

```
a b c d e f g h i j  
1 2 3 4 5 6 7 8 9 10
```

编写替换函数需要注意两个参数的特殊作用

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

‘second’函数在调用时的效果为：

```
> second(t)<-50  
> t  
a b c d e f g h i j  
1 50 3 4 5 6 7 8 9 10
```