

R Project - P1

Storage

OS - environment

R语言是用于统计分析，图形表示和报告的编程语言和软件环境 (35.05) graphical display

an integrated suite of software facilities for data manipulation, calculation and

有一个有效的数据处理和存储工具

an effective data handling and storage facility (readyst - 4) (35.05)

提供一组运算符，用于对数组，列表，向量和矩阵进行计算 (readyst - 4) (35.05)

a suite of operators for calculations on arrays, in particular matrices

提供了一个大型，一致和集成的数据分析工具集合

a large, coherent, integrated collection of intermediate tools of data analysis

提供用于数据处理和直接显示在计算机上或在文档中打印的图形化工具

graphical facilities for data analysis and display either on-screen or on hardcopy

R语言是一种开发良好，简单而有效的编程语言。

包括条件，循环，用户定义的递归函数以及输入和输出工具等

a well-developed, simple and effective programming language

including conditionals, loops, user-defined recursive functions and input and output

注意R语言与Python一样，可是在Command Prompt中打开交互式命令行

同时它一样需要将包含 R.exe 和 Rscript.exe 的文件路径加入系统环境变量 (Windows)

Advanced system settings → Environment Variables → System Variables → Path

(Command) 与 Python 类似，R语言代码可以保存在 .R 后缀的文件中，类似于 .py 文件，

并借助 R 解释器 Rscript 在命令提示符下运行。

如 D:\R> Rscript test.R

注意在 Python 中启动交互式命令行与解释器使用同一个 python.exe 可执行文件

而 R 语言中分别执行 R.exe 和 Rscript.exe

注释 与 Python 类似，R 语言中使用 # 作为注释的提示符。

每一行语句在 # 之后的部分为注释，会被解释器忽略。

如 print("hello world") # comment → [1] "hello world"

注意与 Python 可以使用三重引号表示多行注释不同 ("comment" "comment" "comment")

R 语言不支持多行注释，但存在一种替代方法（不建议使用）

if(FALSE){ "comment" "comment" }，始终为假的条件分支会被解释器忽略

R Project - P2

Java philododony

129 - 3行代码

在R语言中，数据类型的声明与其他编程语言（如C++，Python）不同。
是将变量分配给R对象，并且R对象的数据类型转变为变量的数据类型。

129 - 3行代码

常用的R语言对象有：

向量 (vector)，列表(list)，矩阵(matrix)，数组(array)，因子(factor)，数据帧(frame)

向量 (vector)：在R语言中最基本的数据类型为向量的R对象，用于保存不同类型的元素。

即通过赋值语句将浮点数11.23赋值给变量a。
 $a \leftarrow 11.23$

通过print()语句输出 print(a) → [1] 11.23

可见R语言中向量对象的基本输出格式为下标(index) 值(value)

这种向量的R对象称为原子向量 (atomic vector)

R语言的对象可以通过class()函数返回其类型 (type)

$\text{class}(a) \rightarrow [1] "numeric"$

129 - 3行代码

R语言中共有6种基本原子向量类型 (basic 'atomic' vector type)

类型	type of	mode	storage.mode	范例	print()	class()
逻辑 (浮点)	logical	logical	logical	TRUE, FALSE	[1] FALSE	[1] "logical"
数值	double	numeric	double	1, 0, 11.23	[1] 11.23	[1] "numeric"
整型值	integer	numeric	integer	2L, 0L, -1L	[1] -2	[1] "integer"
复数	complex	complex	complex	1+2i, -1i	[1] 0-1i	[1] "complex"
字符串	character	character	character	'a', 'Hello', "abc"	[1] "Hello"	[1] "character"
原生	raw	raw	raw	48 65 6C 6C 6f	[1] 48 65 6C 6C 6f	[1] "raw"

注意：在赋值时整型必须在结尾加L，即 $a \leftarrow 11L$ ，否则将被视为浮点值。

如果L前含有小数，则抛出警告：contain decimal; using numeric value (强制转换类型)

在print()语句的结果中，l("numeric")和l("integer")是一样的，都是[1] 1

复数类型与Python类似，差别在于用i表示虚部，以及允许虚数表示时出现空格，如 $1 + 2i$ ，并且可以直接作为赋值语句的右值，无需声明class complex()的实例。

字符串型向量与Python类似，可以用单引号或双引号表示，且并不区分字符串

a single element of a character vector is often referred to as a character string

原生类型即Python中的字节串等价，即 class bytes()

即 $\text{charToRaw}("Hello") \rightarrow [1] 48 65 6C 6C 6f$ 与 $b'1\text{x}48\text{x}65\text{x}6C\text{x}6C\text{x}6f'$ 等价

即 $\text{as.raw}(b'1\text{x}48\text{x}65\text{x}6C\text{x}6C\text{x}6f') \text{decode('utf-8')} \rightarrow 'Hello'$

R Project - P3

June 2018

参数 - 参数类型

多元素向量

在R语言中，使用C(args)函数将多个输入组装成一个向量

当输入的所有参数均为原子向量的类型且类型一致时，

R语言将所有的原子向量单向顺序地链接成一个多元向量

如 c(1, 1, 2, 3, 5, 8) → [1] 1 2 3 5 8, class() → [1] "numeric"

c("Hello", "World") → [1] "Hello" "World", class() → [1] "character"

示例：charToRaw("Hello"), charToRaw("Worl")

→ [1] 48 65 6C 6C 6f 57 6f 72 6C 64, class() → [1] "raw"

当输入的参数类型均为原子向量类型，但存在不一致的类型时

R语言会将类型统一为单一类型，除原子向量类型外还有 [1] "list" 类

于是可知，R语言中的向量中的元素应当是同一类型的

首先考虑所有参数均属于原子类型的类型，

在此情况下，R语言会按照一个偏序关系将所有参数解释为其中最高的类型

原子(raw) < 逻辑(logical) < 整型(integer) < 浮点(numeric) < 复数(complex) < 字符(character)

如 cc(charToRaw("Hello"), FALSE) → [1] TRUE TRUE TRUE TRUE TRUE FALSE

cc(FALSE, 11L) → [1] 0 11L class() → "integer" class() → "logical"

cc(1.23, 1+2i) → [1] 1.23+0i 1.00+2i class() → "complex"

cc(1+2i, "Hello") → [1] "1+2i" "Hello" class() → "character"

注意：当输出向量类型为 "numeric" 时

数值的精度，即小数位数，服从输入中精度最高的一个

当输出向量类型为 "complex" 时

实部和虚部的精度分别计算，虚部以所有输入复数虚部精度最高为准

当输出向量类型为 "logical" 时

当输入参数中包含 "logical" 类型时

当解释为 "integer", "numeric", "complex" 中的数值时，TRUE → 1, FALSE → 0

当解释为 "character" 时，TRUE → "TRUE", FALSE → "FALSE"

当输入参数中包含 "raw" 类型时

注意到 "raw" 类型等价于 Python 中的字节串 "bytes"

所以 "raw" 实际上也是以字节为单位存储的，也是以字节为单位解释

当解释为 "logical" 时，每个字节 → TRUE

当解释为 "numeric", "integer", "complex" 时，字节解释为数值

cc(charToRaw("Hello"), 1.23) → [1] 72.00 101.00 108.00 103.00 111.00 1.23

当解释为 "character" 时，直接表示为十六进制形式的字符串

cc(charToRaw("Hello"), "Hello") → [1] "48" "65" "6C" "6C" "6f" "Hello"

R Project - P4

列表 list 在 R 语言中，列表 list 也是 vector。更准确地说是通用向量 (generic vectors)。

列表中包含元素 (element)，每个元素可以包含任意类型的 R 对象 (any type of R object)。

即列表中的元素不必如向量中一样必须为同一类型。

如 `list(c(1, 2), 1+2i, "XYZ", TRUE)` `class() → [1] "list"`

(`list(1+2i)` → `[1] 1+2i` `[2] 1+2i` `[3] "XYZ"` `[4] TRUE`)

除了可以将值装入 list 外，还可以装入函数。

如 `list(sin) → [[1]] function(x) sin`

注意 "list" 本身也是 vector 类型的，所以 list 也可以装入向量。

而 "list" 在向量类型转换过程中，有字符 "character" 与 "list" 列表。

所以如果一个向量赋值中有 list 作为参数，则结果向量类型为 "[1] list"

且向量传入的其他参数保持原值与原有类型，作为结果列表中的元素 FALSE

如 `c(1+2i, list(TRUE, FALSE)) → [[1]] 1+2i [[2]] TRUE [[3]] FALSE`

在生成向量的过程中，如果传入参数有 "list"，则返回也是 "list"

且 R 语言会将所有的参数中的结构打开，将元素依次加入生成的 list

如 `c(1+2i, list(TRUE, FALSE), c("a", "b"))`

`→ [[1]] 1+2i [[2]] TRUE [[3]] FALSE [[4]] "a" [[5]] "b"`

类似于 Python 中传入位置参数和可变参数时使用 * (dereference)

如 `def lst(*args): lst(*[True, False]), 1, *(c('a', 'b'))`

`return args → [True, False, 1, 'a', 'b']`

注意这过程只打开一层结构，如果还嵌套有下层数据结构，则还得拆下来

如 `ccc(1, 1+2i).3 + list(cc(1, 2).3.3, "A")`

`→ [[1]] 1+2i [[2]] 3 [[3]] 1+2i [[4]] 3 [[5]] 1+2i [[6]] 3.3 [[7]] "A"`

在 list 中也可以存在 "list" 元素，而不是如向量一样打开结构。

尽管生成的结果也是 "list" 类型，但是结果有区别。

如 `list(1+2i, list(TRUE, FALSE), c("a", "b"))`

`→ [[1]] 1+2i [[2]] [[1]] TRUE [[3]] [[1]] FALSE [[4]] "a" [[5]] "b"`

这是子列表的索引，形如每一级列表索引的连接。

由此可知，在构造列表，尤其从已有向量或列表构造时，根据需要选择 `c()` 或 `list()`

如果需要展开每个参数并重新封装成一个 list，则使用 `c()`。

如果需要保留每个参数的类型和结构，保持其相对独立性，则使用 `list()`。

(X) 从 do.call 和 do.list 的注释中可以看出它们的实现机制。

R Project - P5

矩阵 (matrix) 在 R 语言中矩阵可以看作一个可以存储不同类型的二维结构。

矩阵是一个二维的 vector，由于 list 也是 vectors 的一种，所以矩阵也可以表现为二维 list。

矩阵通过构造函数 `matrix(source, nrow, ncol, byrow = FALSE)` 生成。

当只传入 source 参数时，`matrix` 会将传入的 vector 或 list 以向地组装成一个矩阵。

如 `matrix(c(1, 2))` → $\begin{bmatrix} 1 \\ 2 \end{bmatrix}$, `matrix(list(1, 2, 2))` → $\begin{bmatrix} 1 \\ 2 \\ 2 \end{bmatrix}$

`length()` → 2, `class()` → [1] "matrix" [2] 2.2

当传入 source 参数，以及 nrow 和 ncol 参数时。

`matrix` 函数会向 $nrow \times ncol$ 矩阵中顺序地填入元素。

如果元素不足以填满矩阵，则返回 source 开头，并从头开始再次调用元素填入矩阵。

如果元素数不是列数的倍数，则会发出警告，但是会完成组装。

如 `matrix(list(1, 2, 3, 3), ncol = 5)` 时

`data length [4] is not a sub-multiple or multiple of the number of columns [5]`

参数 `byrow` 的作用是控制矩阵的组装是以列为基准还是以行为基准。

如 `matrix(list(1, 2, 2, "a", 1+1i), byrow = TRUE)` → $\begin{bmatrix} 1 & 2 & 2 \\ a & 1+1i & \end{bmatrix}$

由于默认值是 FALSE，所以矩阵组装默认是按列的。

由于向量会组装成一个 $n \times 1$ 的矩阵，所以认为 R 语言中的向量通常是纵向量。

而当 `byrow = TRUE` 时，如果出现元素不是行数的因子或倍数时，警告会变为

`data length is not a sub-multiple or multiple of the number of rows`

矩阵的类型由传入的 source 类型决定。

当传入 vector 时，由于 vector 中类型是统一的，所以矩阵中元素的类型也是统一的。

当传入 list 时，矩阵整体表现为 list 类型。

但是其中元素可以表现为不同类型。

如 `matrix(list(1, 2, 3, 3, FALSE, "a", 1+2i), nrow = 2, ncol = 3)` → [1] 1 3 3 [2] 2 FALSE 1+2i → [1] "list"

注意：当只传入 nrow 或 ncol 时，`matrix` 会根据 source 的长度决定 ncol，或 nrow，即 `ncol = ceiling(length(source)/nrow)`, `nrow = ceiling(length(source)/ncol)`

另外 `matrix` 中虽然可以装入函数，但是 print 时不会显示。

如 `matrix(list(1, 2, 2, "a", sin), nrow = 2)` → [1] 1 "a" [2] 2.2

`Print(matrix[2, 2])` → [[1]]

`function(x)`

`Primitive("sin")`

R Project - P6

数组

(array), 在 R 语言中数组可以看作一个可以存储不同类型数据的结构

数组与矩阵 matrix 相似，区别在于矩阵只能是二维的，数组可以是多维的

可以通过 array (source [, dim]) 来生成

如 array(cc1, 2, 3, 4), dim = c(2, 2, 2, 2) →

$$\begin{matrix} \text{cc1} & : & 1,1 & [1,1][2,1] & ; & 2,1 & [1,1][2,1] & , & 1,2 & [1,1][2,1] & , & 2,2 & [1,1][2,1] \\ & | & 1 & 3 & | & 1 & 3 & | & 1 & 3 & | & 1 & 3 \\ [1,1] & & 2 & 4 & | & 2,1 & 2 & 4 & | & 2,2 & 2 & 4 & | & 2,3 & 2 & 4 \end{matrix}$$

dim 参数传入用作决定数组的维度，如 dim = c(2, 2, 2, 2)

参数中元素个数表示 3 数组的维度，而元素本身决定 3 该维度上的元素个数

如 c(2, 2, 2, 2) 定义了一个 $2 \times 2 \times 2 \times 2$ 的数组

dim 参数可以传入 list 且其中元素不全是 "integer" 类型，只要可以解释成整型值

如果传入的值为负数，则抛出错误并终止

如 dim = c(2, -2) → negative length vectors are not allowed

如果传入的值为浮点数，则忽略小数部分。

如 dim = c(2, 2.3) → 2x2 数组

如果传入的值为复数，则采用实部而忽略虚部

如 dim = c(2, 2+1i) → 2x2 数组, imaginary parts discarded in coercion

如果传入的值为字符，则尝试将字符串表示为数字 "numeric" 类型

如 dim = c(2, "2.9") → 2x2 数组

但是如果无法表示成数字，则抛出错误

如 dim = c(2, "z") → NAs introduced by coercion

注意 dim 参数表示的维度是从左至右排列的，即 -, -, 3rd, 4th, ... 列标

即 (行标, 列标, 3rd, 4th, ...), 如显示时 行标 元素

一方面解释了 $2 \times 2 \times 2 \times 2$ 数组显示时，-, -, 2, 1 先于 -, -, 1, 2 出现

另一方面如果不传入 dim 参数，则组装形如纵向的一维数组

source 参数至少可以传入向量、列表、矩阵，且对于 array() 函数处理方式类似

都是对于 source 中顺序提取元素填入已定义好尺寸的 array，直至装满

且与 matrix 不同，即使元素个数不是行/列数的因子和倍数也不会提示

注意当生成的数组是一维的，其类型依旧是数组，但显示 (print()) 与一维向量和 list 一样

如 array(list(1, FALSE, "a")), → [[1]] 1 [[2]] FALSE [[3]] "a"

list(1, FALSE, "a"), → [[1]] 1 [[2]] FALSE [[3]] "a"

但是前者 class() → [1] "array", 而后者 class() → [1] "list"

但如果数组是二维的，则会直接转换成矩阵类型

array(list(1, FALSE, "a"), dim = c(2, 2)) → [1, 1] 1 "a"

注意：数组总是纵向填装的

class() → [1] "matrix" [2,] FALSE 1

R Project - P7

因子

(factor). 在R语言中，用于描述具有有限规模的值的项目

可以使用 factor() 函数生成

如 `factor(c(1, 1, 2, 3, 5, 8))` → `class()`: [1] factor print() →

因子具有 "factor" 类型和一个 "levels" 属性

levels 属性表示了 factor 对象中的不同元素

可以通过 nlevels() 函数返回不同元素个数

因子还具有可选属性 "contrasts" 在建模函数中用于控制参数化

control the parametrisation used when used in modeling function

因子可以是纯名义的或者拥有有序的分类，可以被定义为 vector 类型

may be purely nominal or have ordered categories

defined as such and have a class vector

在 R 语言中，目前因子的实现是通过一个整数数组来指定实际的 level

以及第二个数组，将 names 映射到整数

currently implemented using integer array to specify the actual levels

a second array of names that mapped to integer

数据帧

(data frame) 在 R 语言中，data frame 用于描述表格型数据，类似于数据库中的 relation

可以使用 data.frame() 函数生成

如 `data.frame(id = c(101, 102, 103), name = c("a", "b", "c"), pass = c(TRUE, FALSE, TRUE))`

→ [1] "data.frame" 3 101 "a" TRUE
2 102 "b" FALSE
3 103 "c" TRUE

注意 data.frame() 函数会将传入的参数对应位置的元素合并成一行，或者一个 n 元组

所以传入参数时，参数长度应该一致，否则抛出 argument imply different number of rows

另外，matrix 也可以作为参数传入，如 ..., other = ... → ... other.1 other.2

但是要求行数必须一致 matrix(c(1, 2, 3, 4, 5, 6), nrow=3, ncol=2)

会出现和上述一样的错误

另外，列表也可以作为参数传入，且如 ..., other = ... → ... id other.1 other.2

但是不受行数要求约束

而是对于 list 中每个元素的长度约束

如果只有一个元素则整个填充为该元素

但是如果某个元素行数是其他行数的倍数，则扩展至与最大行数

但是如果是

"numeric" 或 "logic"

则等价于用列向量 vector

字符串 "character"

返回 factor

可以通过形如 `frame[, col]` 提取一列的值，返回向量，如 `frame[, 1]` → [1] 101 102 103

通过形如 `frame[row,]` 提取一行的值，返回 frame，如 `frame[1,]` → 1 101 "a" TRUE

通过形如 `frame$name` 提取某一列信息，返回 factor，如 `frame$name` → [1] a b c

levels : a b c

R Project - P8

变量

在 R 语言中，变量可以绑定原子向量、原子向量或 R 对象的组合。有效的变量名应由字母、数字、下划线或点(.) 组成。

注意对于变量名的限制为：

变量的第一个字符或者为字母，如 var-name.2

或者为点，且点后一位字符必须不是数字。

例如：var_name.2, .var_name2

注意，当非法变量名为出现不合法字符，如 var-name%#%

或以下划线开头时，如 _var_name

则 R 语言抛出 unexpected input 错误。

当非法变量名为以数字开头，如 2Var-name

或在点后出现数字，如 .2Var-name

则 R 语言抛出 unexpected symbol 错误。

变量可以通过 print() 函数或 cat() 函数输出到显示终端。

但是注意 print() 函数只能输出一个变量。

而其字符串参数会被忽略。

如 print(1, 2, 3) → [1] 1

而 print() 的控制参数则要求必须以关键字参数形式传入。

如 print(table(c(0, 1, 2), c(0, 2, 4)), zero.print = TRUE)

zero.print = TRUE

cat()

cat() 函数可用于输出多个变量，形式上与 Python 一致。

但是注意 cat() 函数并不会在结尾默认输出一个换行符。

另外 cat() 函数在每个传入参数的输出间插入一个空格。

如 > cat("1\n", "2\n", "3") → 1 2

3> (prompt)

在 R 语言中，变量本身不具有类型，而是仅反映其指向的对象的类型。

所以 R 语言与 Python 一样是动态语言，即可向同一变量名绑定不同类型的值。

ls()
在 R 语言中，可以通过 ls() 函数返回当前域或变量名的列表。
class(ls()) → [1] "character"
注意以点开头的变量被认为是隐藏，需要对 ls() 函数传入关键字参数 all.names = TRUE。

rm()
在 R 语言中可以通过 rm() 函数删除当前域中的指定变量，如 rm(arg1)

也可以通过传入多个参数删除多个变量，如 rm(arg1, arg2, ...)

也可以通过将 ls() 作为关键字参数传入以删除当前域所有变量，如 rm(list = ls())

R Project - Pg

运算符

在R语言中，提供的算术运算符包含基本的算术运算

如 + (加法), - (减法), * (乘法), / (除法), ^ (幂次)

% (取模), %/ (取余)

在R语言中，通常数值都以 "numeric" 的形式保存。

在符号运算符定义的情况下，以上运算符均支持非整数的运算

其中 %% (取模), %/ (取余) 运算遵循以下定义

$$\begin{cases} a \% \% b = \lfloor a/b \rfloor, & \text{当 } b \neq 0 \\ a \% \% b = a - (a \% \% b) * b = a - \lfloor a/b \rfloor b \end{cases}$$

注意这个定义对所有实数有效

注意这个定义与 Python 和 Haskell - 致，但是 C++ 和 Racket 则不同

如 [1] -5 -4 -3 -2 -1 0 1 2 3 4 5

→ %/ 3 [1] -2 -2 -1 -1 -1 0 0 0 1 1 1

→ %% 3 [1] 1 2 0 0 1 2 0 1 2 0 1 2

取余 → %/ -3 [1] 1 2 1 0 0 1 0 0 1 0 1 2

→ %% -3 [1] -2 -1 0 0 -2 -1 0 -2 -1 0 -2 -1

→ %/ -2.5 [1] 2 1 0 0 0 0 0 0 1 1 -2 -2 -2

取模 → %% -2.5 [1] 0.0 -1.5 -0.5 -2.0 -1.0 0.0 -1.5 -0.5 -2.0 -1.0 0.0

在R语言中，算法运算符可以看作是自带 map 功能的

首先对于 向量与立即数的二元运算

可以看作是先向二元运算传入参数立即数并返回一个 partial function

再将这个函数 map 到向量、矩阵、数组上，但是无法直接 map 到列表上

注意对于不服从交换律的运算要明确运算顺序

如 mat ← matrix(c(1, 2, 3, 4, 5, 6), nrow=2, ncol=3)

mat ^ 2 → [1,] [2,] [3,] [1,] [2,] [3,]
[1,] 1 4 16 [1,] 2 8 32
[2,] 4 16 64

其次是对于向量、矩阵、数组之间的算术运算符 map 运算

在类型相同且尺寸相同的情况下，会一一对应地进行运算

如 c(1, 2, 3) ^ c(2, 3, 4) → [1] 1 8 27

如果类型不同，则只有向量可以和其他两种类型 map 运算

如果尺寸不同，则要求一个向量的元素数是另一个向量的倍数

或者矩阵、数组的元素数是向量元素数的倍数，且 map 顺序为先列再行

如 , , 1 [1,] [2,] [3,] , , 2 [1,] [2,] [3,] ^ c(1, 2, 3) * [1,] [2,] [3,] ^ [1,] [2,] [3,]

[1,] 1 3 1 9 [1,] 3 9 27 [1,] 1 27 1 [1,] 3 1 9
[2,] 2 4 2 [2,] 4 2 4 [2,] 4 4 8 [2,] 16 2 64

R Project - P10

关系运算符

在R语言中，提供了关系运算符，用于比较两个元素的值

包括 LT(<), LE(<=), EQ(==), NE(!=), GE(>=), GT(>)

关系运算符与算术运算符一样，会自动 map 到对应元素

如 $\begin{bmatrix} 1, 1 \\ 2, 2 \end{bmatrix} < \begin{bmatrix} 1, 2, 3 \end{bmatrix} \rightarrow \begin{bmatrix} \text{FALSE} \\ \text{FALSE} \end{bmatrix}$

关系运算符可以作用于不同类型的元素之间

其计算方式是先按照类型转换的偏序角解释为最高类型，再进行比较

对于解释为 "complex" 的情况，会报出错误信息

如 $\text{cc}(1, 0, 1000) < \text{cc}(1+1i, 1+1000i)$

抛出 invalid comparison with complex values

对于解释为 "character" 的情况，会按照字典序进行比较

注意：在R语言中字符串实际上视为长度为1的字符串

如 $\text{cc}'x', 'y', 'aa' < \text{cc}'a', 'b', 'c'$ → [1] FALSE FALSE TRUE

当用于比较的其中一个向量类型为 [1] "logical" 时

如果最终解释为 "numeric"，则 TRUE 视为 1，FALSE 视为 0

如 $\text{c}(\text{TRUE}, \text{FALSE}, \text{FALSE})$

$= \text{cc}(1, 1, -1)$ → [1] TRUE, FALSE, FALSE

$< \text{cc}(1, 1, -1)$ → [1] FALSE, TRUE, FALSE

$> \text{cc}(1, 1, -1)$ → [1] FALSE, FALSE, TRUE

如果最终解释为 "character"，则 TRUE 视为 "TRUE"，FALSE 视为 "FALSE"

如 $\text{c}(\text{TRUE}, \text{FALSE}, \text{FALSE})$

$> \text{cc}'s', 'y', 'aa'$ → [1] TRUE, FALSE, TRUE

注意：在R语言的字符串间的比较中，

比较运算并不遵循一般意义上的基于 ASCII 码的比较

如原本在 ASCII 码顺序中，大写字母 < 'l', 'j' < 小写字母

而在R语言中表现为，符号小于字母和数字

$\text{cc}'{', 'j'} < \text{cc}'A', 'l'$ → [1] TRUE TRUE

字符串只遵循字典顺序，而小写字母 < 大写字母

$\text{cc}'false', 'False', 'FALSE'$

$= \text{cc}(\text{FALSE}, \text{FALSE}, \text{FALSE})$ → [1] FALSE FALSE TRUE

$< \text{cc}(\text{FALSE}, \text{FALSE}, \text{FALSE})$ → [1] TRUE TRUE FALSE

另外所有的数字均小于字母

于是推测 R 语言的字符串顺序为

符号 < '0', '1', ..., '9' < 'a', 'A', 'b', 'B', ..., 'z', 'Z'

R Project - P11

逻辑运算符 在R语言中，提供了两套逻辑运算符。
向量化(vectorized)的：& (按位与), | (按位或), ! (按位非)
非向量化(not vectorized)的：&& (与), || (或)
这两者的区别在于是针对元素的还是针对整体的。
向量化的&, |, ! 与算术运算符和关系运算符类似
即自动地匹配两个运算向量中的元素。
如 $\text{!c}(0, 1, 2) \rightarrow [1] \text{ TRUE } \text{ FALSE } \text{ FALSE }$
 $\text{c}(0+0i, 0+1i, 1+0i, 1+1i) \& \text{c}(\text{TRUE}, \text{FALSE})$
 $\rightarrow [1] \text{ FALSE } \text{ FALSE } \text{ TRUE } \text{ FALSE }$
而非向量化的&&, || 则只取两个运算向量中的第一个元素, 而忽略其他元素
如 $\text{c}(0+0i, 0+1i, 1+0i, 1+1i) \&\& \text{c}(\text{TRUE}, \text{FALSE})$
 $\rightarrow [1] \text{ FALSE }$, 即 $\text{c}(0+0i) \&\& \text{c}(\text{TRUE})$
特别注意, 非运算“!”只有向量化的而没有非向量化的。

另外逻辑运算符仅支持“logical”, “numeric”和“complex”类型
如当用非运算 $! \text{c}("a", "b", "c")$ 时
抛出错误 invalid argument type
当用按位与运算 $\text{c}("a", "b", "c") \& \text{c}(\text{TRUE})$ 时
抛出错误 operations are only possible for numeric, logical or complex types
当用与运算 $\text{c}("a", "b", "c") \&\& \text{c}(\text{TRUE})$ 时
抛出错误 invalid 'x' type in 'x && y'
当“numeric”类型用于逻辑运算符时,
整型值 0L, 浮点数 0 解释为 FALSE, 其他值解释为 TRUE
当“complex”类型用于逻辑运算符时
0+0i 解释为 FALSE, 其他复数解释为 TRUE

赋值运算符 在R语言中，提供了3种赋值运算符，左赋值(<-)，右赋值(→)，贝赋值(=)
其中 左赋值(<-) 和 右赋值(→) 看作单一的赋值运算
即 允许以 相同 的方向连续运行，或是同一个运算符连续执行。
如 $\text{vec}_1 <- \text{vec}_2 <- \text{c}(1, 2, 3) \rightarrow [1] 1 \cdot 2 \cdot 3$
 $\text{vec}_3 <- \text{c}(1, 2, 3) \rightarrow \text{vec}_4 \rightarrow \text{vec}_5 \rightarrow [1] 123, [1] 123, [1] 123$
而如 $\text{vec}_6 <- \text{c}(1, 2, 3) \rightarrow \text{vec}_7 <- \text{c}(4, 5, 6)$
则会解释为 $(\text{vec}_7 <- \text{c}(1, 2, 3)) <- \text{c}(4, 5, 6)$
抛出错误 could not find function "<-<-"

R Project - P12

赋值命令

在 R 语言中，赋值运算符 (=) 的运算方式与一般编程语言中的赋值命令类似，即采用 $\text{left value} = \text{right value}$ 的形式。将 right value 中包含的表达式计算值绑定到 left value 所指向的变量名，并支持自右向左的连接赋值。

如 $\text{vec}_3 = \text{vec}_9 = \text{c}(1, 2, 3) \rightarrow [1] 1 2 3$

也可以与左赋值 (<-) 和右赋值 (->) 混合使用。

即左赋值与右赋值作为 right value 中的一部分。

如 $\text{vec}_{10} = \text{vec}_{11} <- \text{c}(1, 2, 3) \rightarrow [1] 1 2 3, [1] 1 2 3$

$\text{vec}_{12} = \text{c}(1, 2, 3) \rightarrow \text{vec}_{13} \rightarrow [1] 1 2 3, [1] 1 2 3$

但是不能作为 left value 部分。

如 $\text{vec}_{14} <- \text{c}(1, 2, 3) = \text{c}(4, 5, 6)$ 出错：
"could not find function '<-<-'"

序列生成

在 R 语言中，冒号运算符 (:) 用于生成一个数值序列。

与 Python 中的 range(start, end) 类似。

即形如 $\text{start}: \text{end}$ 的表达式可生成 $\text{start}, \text{start}+1, \dots, \text{end}$ 的序列。

注意：当 start, end 均为整型值时

生成序列即 $\text{start}, \text{start}+1, \dots, \text{end}$

如 $1:11 \rightarrow [1] 1 2 3 4 5 6 7 8 9 10 11$

注意在 R 语言中，(:) 固定步长为 1，即每个元素为前一个元素 +1

另外截止区间为闭区间，即 $[\text{start}, \text{end}]$

但在 Python 中为右侧开区间 $[\text{start}, \text{end})$

当 start 为 "numeric" 时，则依旧保持 +1，不论 end 是什么类型

如 $0.5:11 \rightarrow [1] 0.5 1.5 2.5 3.5 4.5 5.5 6.5 7.5 8.5 9.5 10.5$

当 start 为 "complex" 时，则只在实部 +1，虚部则需分情况

如果 end 的类型也是 "complex"，R 将 start 实部递增

如 $0+0i:5+0i \rightarrow [1] 0+0i, 1+0i, 2+0i, 3+0i, 4+0i, 5+0i$

但是特别注意，当 Start 虚部不为 0 时

会被 end 的虚部覆盖。如果 end 为 "numeric" 则会忽略虚部

并抛出警告：imaginary parts discarded in coercion

如 $0+2i:5+1i \rightarrow [1] 0+1i, 1+1i, 2+1i, 3+1i, 4+1i, 5+1i$

如 $0+2i:11 \rightarrow [1] 0 1 2 3 4 5 6 7 8 9 10 11$

当 start 或 end 为 "logical" 时，则会解释为 0 (FALSE) 或 1 (TRUE)

R Project - P13

匹配运算符 在R语言中，匹配运算符 (%in%) 用于查询左运算数是否在右运算数中。其执行逻辑与 Python 中查询是否为集合元素的关键字 in 类似。

如 $1 \%in\% c(1, 2, 3, 4) \rightarrow [1] \text{ TRUE}$

$1 \%in\% c(1, 2, 3, 4) \rightarrow [1] \text{ FALSE}$

注意：匹配运算符是向量化的，即对于左运算数的每个运算作一次运算。

如 $3:5 \%in\% c(1, 2, 3, 4) \rightarrow [1] \text{ TRUE} \quad [2] \text{ TRUE} \quad [3] \text{ FALSE}$

当匹配运算符作用于 ~~非向量左操作数~~ 时，会破坏左运算数原有的结构，而统一返回向量作为结果。

如 $\begin{bmatrix} 1, 1 \\ 2, 1 \end{bmatrix} \%in\% \begin{bmatrix} 1, 2, 3 \\ 2, 4 \end{bmatrix} \rightarrow [1] \text{ FALSE} \quad [2] \text{ TRUE} \quad [3] \text{ FALSE} \quad [4] \text{ TRUE}$

$\begin{bmatrix} 1, 1 \\ 2, 1 \end{bmatrix} \%in\% \begin{bmatrix} 1, 2 \\ 2, 8 \end{bmatrix} \rightarrow [1] \text{ TRUE} \quad [2] \text{ FALSE}$

$\begin{bmatrix} 1, 1 \\ 2, 1 \end{bmatrix} \%in\% \begin{bmatrix} 1, 2 \\ 2, 8 \end{bmatrix} \rightarrow [1] \text{ TRUE} \quad [2] \text{ FALSE}$

$\begin{bmatrix} 1, 1 \\ 2, 1 \end{bmatrix} \%in\% \begin{bmatrix} 1, 2 \\ 2, 8 \end{bmatrix} \rightarrow [1] \text{ TRUE} \quad [2] \text{ FALSE}$

$\rightarrow [1] \text{ TRUE} \quad [2] \text{ TRUE} \quad [3] \text{ FALSE} \quad [4] \text{ TRUE}$

$[1] \text{ TRUE}$

$\text{list}(1:3, 4, 5) \%in\% \begin{bmatrix} 1, 3 \\ 2, 4 \end{bmatrix} \rightarrow [1] \text{ FALSE} \quad [2] \text{ TRUE} \quad [3] \text{ FALSE}$

$\rightarrow [1] \text{ FALSE} \quad [2] \text{ TRUE} \quad [3] \text{ FALSE}$

当左操作数或右操作数为列表时，列表的每一项作为一个独立的元素看待。

如 $1:3 \%in\% \text{list}(1:3, 4:7, 8:11) \rightarrow [1] \text{ FALSE} \quad [2] \text{ FALSE} \quad [3] \text{ FALSE}$

$\text{list}(1:3, 4, 5) \%in\% \text{list}(1:3, 4:7, 8:11)$

$\rightarrow [1] \text{ TRUE} \quad [2] \text{ FALSE} \quad [3] \text{ FALSE}$

当查询元素是否在数据框 data.frame 内时

如果声明调用列，则等同于查询向量。

$\text{list}(1:3, 4, 5) \%in\% \text{data.frame}(x=1:3, y=4:6) \$ y$

$\rightarrow [1] \text{ FALSE} \quad [2] \text{ TRUE} \quad [3] \text{ TRUE}$

如果不声明，则类似于列表将每一列作为一个单一元素看待。

$\text{list}(1:3, 4, 5) \%in\% \text{data.frame}(x=1:3, y=4:6)$

$\rightarrow [1] \text{ TRUE} \quad [2] \text{ FALSE} \quad [3] \text{ FALSE}$

矩阵乘法 在R语言中，矩阵乘法 (%*%) 用于两个矩阵的乘法。

如 $\begin{bmatrix} 1, 1 & 2 \\ 2, 1 & 4 \end{bmatrix} \%*% \begin{bmatrix} 1, 2, 3 \\ 2, 4, 6 \end{bmatrix} \rightarrow [1] \text{ 7} \quad [2] \text{ 15} \quad [3] \text{ 23}$

如果大小不对
则抛出
non-comformable
arguments

注意：如果不是矩阵或向量，或类型不是数字或复数。

抛出错误： requires numeric/complex matrix/vector arguments

特别注意：如果有相同数量元素的向量或数组，则类似于求 sum of product。

R Project - P14

29 - send off

控制结构 (control structure) 在 R 语言中，顺序地计算语句 (sequentially evaluate statements)

语句间可以用分号 (;) 或换行分隔，如 $x \leftarrow 0; x + 5 \rightarrow [1] 5$

当运算得到一个语法上完整的语句，即计算值并返回

syntactically complete statement evaluated and value returned

if/else 结构，从语法上看，R 语言的 if/else 结构与 C++ 类似

但是 if/else 本身可以视为一个表达式，并用于赋值

如 $x \leftarrow \text{if} (11 \% 2 == 0) 11 \% 2 \text{ else } 11 * 3 + 1 \rightarrow 34$

注意这个赋值语句与 Python 类似

另外注意在 R 语言中，最后一个 else 语句是可省略的

如果省略，则 if/else 在此处的计算值为 NULL

switch 结构，在 R 语言中，switch 结构的基本语法为 switch (expr, ...) 其中 (...) 表示

(...) 中的参数会组装为一个 list，这个 list 可以是命名的 (named)

如果 expr 计算结果为一个在 1 到 list.length 之间的整数

则返回 list 中对应的值，如 switch (2, 'a' = 1, 3, 'b' = 5) $\rightarrow 3$

超出范围时返回 NULL，如 switch (5, 'a' = 1, 3) $\rightarrow \text{NULL}$

如果 expr 为一个字符串向量，则作为 key 在 list 中查找对应的值

返回值

注意，list 必须是命名的，但可以有至多一个未命名元素作为查找失败的默认

如果没有对应的 key 也没有提供 default，则返回 NULL

repeat 结构，在 R 语言中提供的循环结构之一，基本语法为 repeat {statements}

注意这个结构实际上等同于 C++ 中的 while (true) { ... }

所以在使用时应明确地使用 break 或 return

以防 repeat 结构陷入死循环

while 结构，在 R 语言中提供的循环结构之一，基本语法结构为 while (statement) {statements}

其运算逻辑与 C++ 和 Python 中的 while 循环一致

for 结构 在 R 语言中提供的循环结构之一，基本语法结构为 for (name in vector) {statements}

其运算逻辑与 Python 中的 for 循环一致

但是特别注意，R 语言的 for 循环有副作用 (side-effect)

即在循环体内的 name 会继续存在，且值等于 vector 中最后一个元素

另外，在 R 语言中提供 next 命令用于跳到循环的下一次迭代，与 C++ 的 continue 类似