Linguagem R de programação Estatística

Professor Eduardo Monteiro de Castro Gomes

A forma mais simples de utilização do R é como uma calculadora para realizar operações básicas como:

```
2 + 2
```

```
## [1] 4
```

ou

```
1348.75 / 7
```

```
## [1] 192.6786
```

e conjuntos de operações

```
(783 - 139)^2 / 5
```

```
## [1] 82947.2
```

Essa utilização simplória começa a se tornar vantajosa, em relação a uma simples calculadora, a partir do momento em que é possível armazenar o resultado das operações realizadas para reutilização em outras operações.

No exemplo seguinte temos o valor de um determinado produto, 750 reais, ao qual será aplicado um desconto, 12%, e o valor final calculado e salvo em um objeto chamado *preco_final*.

```
preco_final <- 750 * .88
preco_final
```

```
## [1] 660
```

Para fazer a atribuição de um objeto utiliza-se "<-" determinando o valor no lado esquerdo como nome do objeto representado no lado direito. É importante notar que a linguagem faz distinção entre letras minúsculas e maiúsculas. Os nomes podem ser formados por letras, números, "_" e ".", devendo ser iniciados por letras ou por pontos desde que não sejam seguidos por números.

```
nome_valido1 <- 10
.outro_valido <- 10
```

Voltando ao exemplo do produto com desconto, suponha que tem-se o interesse em parcelar o pagamento do produto em 3 parcelas e pode-se utilizar o nome do objeto em que foi armazenado o preço final do produto com o desconto para calcular o valor de cada parcela.

```
preco_final / 3
```

```
## [1] 220
```

Vetores

É natural considerar que as operações que serão realizadas com auxílio da linguagem R consideram grandes números de observações. Uma primeira extensão que vamos considerar aos objetos númericos que vimos no exemplo anterior é pela utilização de vetores que permitem armazenar conjuntos de valores.

A função para criação dos vetores é *c()* em que a letra c é utilizada como abreviação de concatenação que será realizada entre os objetos para serem agrupados em um único objeto.

```
precos <- c(750, 822, 300, 15)
precos
```

```
## [1] 750 822 300  15
```

e temos assim em um único objeto, chamado precos, um vetor que contem os valores 750, 822, 300 e 15. Note que a função *c()* pode fazer tambem a concatenação entre vetores.

```
vetor1 <- c(1,2,3,4)
vetor2 <- c(100,200,300)
vet_concatenado <- c(vetor1,vetor2)
vet_concatenado
```

```
## [1]   1   2   3   4 100 200 300
```

e **vet_concatenado** é o vetor resultante da concatenação entre os vetores *vetor1* e *vetor2*.

Funções úteis para criação de vetores numéricos

Algumas funções comumente utilizadas para criação de vetores com valores numéricos são exemplificadas a seguir:

```
1:10
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
10:1
```

```
##  [1] 10  9  8  7  6  5  4  3  2  1
```

```
-5:3
```

```
## [1] -5 -4 -3 -2 -1  0  1  2  3
```

```
1.5:8
```

```
## [1] 1.5 2.5 3.5 4.5 5.5 6.5 7.5
```

o operador **:** cria sequências de números com início no valor à esquerda incrementando ou decrementando uma unidade até um limite determinado pelo valor à direita do operador.

A função *seq* tem comportamento semelhante mas permite a determinação do tamanho do incremento ou decremento, ou a determinação do números de elementos igualmente espaçados desejados dentro dos limites determinados, conforme os exemplos:

```
seq(from = 1, to = 10, by = .5)
```

```
##  [1]  1.0  1.5  2.0  2.5  3.0  3.5  4.0  4.5  5.0  5.5  6.0  6.5  7.0  7.5
## [15]  8.0  8.5  9.0  9.5 10.0
```

```
seq(from = 1, to = 10, length.out = 25)
```

```
##  [1]  1.000  1.375  1.750  2.125  2.500  2.875  3.250  3.625  4.000  4.375
## [11]  4.750  5.125  5.500  5.875  6.250  6.625  7.000  7.375  7.750  8.125
## [21]  8.500  8.875  9.250  9.625 10.000
```

A função *rep* permite criar um vetor a partir da repetição de um elemento ou vetor, permitindo determinar o número de repetições para cada elemento de forma sequencial ou não.

```
rep(c(1,2,3), times = 3)
```

```
## [1] 1 2 3 1 2 3 1 2 3
```

```
rep(c(10,20,30), each = 3)
```

```
## [1] 10 10 10 20 20 20 30 30 30
```

```
rep(c(10,20,30), times = c(1,2,3))
```

```
## [1] 10 20 20 30 30 30
```

É possível gerar números aleatórios a partir de diferentes distribuições de probabilidade. As funções para geração de números aleatórios tem seus nomes formados pela letra *r*, de random, seguido pelo nome ou abreviação do nome da distribuição de probabilidade. A seguir são apresentados exemplos de números gerados pelas distribuições Normal e Poisson.

```
rnorm(n = 10, mean = 100, sd = 25)
```

```
##  [1] 170.86921  89.71280  91.54541 106.59859 100.15275 112.06544 115.13473
##  [8]  97.46172  96.46007 125.94947
```

```r
rpois(n = 15, lambda = 500 )
```

```
##  [1] 513 527 522 527 521 511 505 521 515 468 508 506 452 540 464
```

Tipos de elementos

Até este ponto do texto foram considerados apenas exemplos em que as variáveis utilizadas foram todas numéricas, mas é bastante comum em linguagens de programação que se trabalhe com outros tipos de variáveis. Estaremos interessados aqui, alem das variáveis númericas, nas variáveis do tipo lógico e do tipo caracter.

- Variáveis lógicas

As variáveis do tipo lógico podem assumir apenas dois valores: TRUE e FALSE, representando verdadeiro e falso. Esse tipo de variável lógica será de grande importância e é resultante principalmente de operações de comparação entre elementos do tipo:

```
3 > 5
```

```
## [1] FALSE
```

```
3 < 5
```

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

```
3 >= 5
```

```
## [1] FALSE
```

```
3 <= 5
```

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

```
3 == 5
```

```
## [1] FALSE
```

```
3 == 3
```

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

Um vetor de elementos lógicos pode ser definido pela criação dos elementos conforme o exemplo a seguir:

```
novo_vetor <- c(TRUE, T, FALSE, F)
```

note que a criação de elementos lógicos com valor verdadeiro pode ser feita utilizando a palavra completa *TRUE* ou apenas a primeira letra *T*, sempre com letras maiúsculas. Os elementos com valor falso, de forma análoga podem ser criados com *FALSE* ou apenas *F*.

E vetores com elementos lógicos podem ser criados apartir de operações de comparação entre vetores

```
limite <- 5
notas_alunos <- c(8, 6, 3, 7, 4, 10)
aprovacao <-  notas_alunos >= limite
aprovacao
```

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

e nesse exemplo supondo que temos um vetor com as notas de alunos e a nota limite inferior para aprovação seja 5 então o vetor aprovação traz o resultado lógico indicando para as respectivas notas se o aluno foi aprovado

- Variáveis caracter

As variáveis do tipo caracter são utilizadas para armazenar palavras ou textos e devem ser definidas com a utilização de áspas “”

```
servidores <- c("Ana", "Pedro", "Carolina")
servidores
```

```
## [1] "Ana"      "Pedro"    "Carolina"
```

Um objeto que pode ser útil em algumas aplicações para criação de vetor com caracteres em sequência

```
letters
```

```
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q"
## [18] "r" "s" "t" "u" "v" "w" "x" "y" "z"
```

```
LETTERS
```

```
##  [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q"
## [18] "R" "S" "T" "U" "V" "W" "X" "Y" "Z"
```

Operações entre vetores

Uma das grandes vantagens da linguagem R de programação estatística é sua implementação de operações entre vetores. Vejamos um exemplo análogo ao visto anteriormente em que tinha-se interesse em calcular o preço final de um produto após aplicar um desconto de 12%. Considere que temos o vetor *precos* que armazena o preço de diferentes produtos e desejamos calcular o preço final de cada um desses produtos após a aplicação do desconto. Pode-se realizar essa operação de forma simples na linguagem R com sua implementação vetorizada das operações

```
precos
```

```
## [1] 750 822 300  15
```

```
precos * .88
```

```
## [1] 660.00 723.36 264.00  13.20
```

e dessa forma foram calculados os preços finais dos quatro produtos com uma única operação. Nesse exemplo foram realizadas as operações entre um vetor e um escalar(vetor de tamanho 1), mas as operações vetorizadas em R permitem também que as operações sejam realizadas entre dois vetores elemento a elemento.

```
vetorA <- c(1,2,3,4)
vetorB <- c(10,20,30,40)
vetorA + vetorB
```

```
## [1] 11 22 33 44
```

No exemplo acima ambos os vetores tinham a mesma dimensão com 4 elementos cada. As operações entre vetores não estão restritas a vetores com mesma dimensão e a linguagem R utiliza-se de um procedimento chamado reciclagem para a realização de operações entre vetores de dimensões diferentes.

Os exemplos abaixo apresentam a forma em que a reciclagem é utilizada

```
v1 <- c(100,200,300,400)
v2 <- c(1,2)
v1 +  v2
```

```
## [1] 101 202 301 402
```

Note que ao realizar a soma entre os elementos dos vetores o primeiro elemento do vetor *v1* é somado ao primeiro elemento do vetor *v2* e o segundo elemento do vetor *v1* é somado ao segundo elemento do vetor *v2*. Ao realizar a soma para o terceiro elemento do vetor *v1* o vetor *v2* não tem mais elementos para a soma e dessa forma o vetor *v2* é reciclado e seu primeiro elemento é utilizado na soma com o terceiro elemento do vetor *v1* e posteriormente o quarto elemento do vetor *v1* é somado ao próximo elemento do vetor reciclado *v2* que é seu segundo elemento.

A linguagem utiliza por padrão esse procedimento de reciclagem, de forma que em nosso exemplo em que o desconto foi aplicado ao preço de diferentes produtos com uma única operação o valor do desconto foi reciclado na multiplicação pelo preço de cada um dos produtos.

Dependendo das dimensões dos vetores considerados na reciclagem, quando as operações de reciclagem não reutilizam completamente o vetor reciclado, uma mensagem de aviso é apresentada ao usuário para alertar sobre uma falta de conformidade entre as dimensões dos vetores utilizados. O exemplo a seguir ilustra esse aviso gerado:

```
V1 <- c(100,200,300)
V2 <- c(1,2)
V1 + V2
```

```
## Warning in V1 + V2: longer object length is not a multiple of shorter
## object length
```

```
## [1] 101 202 301
```

Note que o primeiro elemento de *V1* foi somado ao primeiro elemento de *V2* e o segundo elemento de *V1* foi somado ao segundo elemento de *V2*. Para somar o terceiro elemento de *V1* o vetor *V2* precisou ser reciclado de forma que esse terceiro elemento de **V1** foi somado ao primeiro elemento do vetor reciclado *V2*. O próximo elemento do vetor reciclado *V2* não é utilizado, uma vez que não existem mais elementos do vetor *V1* para ser somado. A operação de soma é realizada com a utilização de reciclagem parcial do vetor *V2* mas uma mensagem de aviso é gerada para alertar o usuário.

Indexação de vetores

Os elementos de um vetor podem ser acessados pela posição em que estão. Quando se tem interesse em acessar um determinado elemento dentro de um objeto utiliza-se [] após o nome do elemento e dentro desse operador deve se indicar os elementos de interesse.

Vamos considerar o seguinte vetor para exemplo:

```
set.seed(123)
numeros <- rpois(n = 6, lambda = 300)
numeros
```

```
## [1] 290 320 270 302 329 307
```

se tivermos interesse no número 270 que está na terceira posição pode-se acessa-lo conforme o exemplo:

```
numeros[3]
```

```
## [1] 270
```

pode-se indicar um vetor com os índices de interesse e no exemplo abaixo iremos selecionar os elementos que estão nas posições 2 e 4 do vetor *numeros*

```
numeros
```

```
## [1] 290 320 270 302 329 307
```

```
numeros[c(2,4)]
```

```
## [1] 320 302
```

Alternativamente à escolha dos elementos que se deseja acessar em um vetor pode-se indicar utilizando o sinal **-** os elementos que não devem ser acessados. No exemplo abaixo vamos acessar todos os elementos do vetor *numeros* menos o segundo elemento

```
numeros
```

```
## [1] 290 320 270 302 329 307
```

```
numeros[-2]
```

```
## [1] 290 270 302 329 307
```

e de forma análoga pode-se selecionar um vetor de índices que não serão acessados e no exemplo serão acessados todos os elementos do vetor *numeros* exceto os elementos com índices 1 e 4

```
numeros
```

```
## [1] 290 320 270 302 329 307
```

```
numeros[-c(1,4)]
```

```
## [1] 320 270 329 307
```

A indexação dos elementos de um vetor também pode ser realizada a partir dos elementos lógicos. Suponha que tem-se interesse em acessar os dois primeiros e o último elemento do vetor *numeros*

```
numeros
```

```
## [1] 290 320 270 302 329 307
```

```
numeros[c(T,T,F,F,F,T)]
```

```
## [1] 290 320 307
```

o vetor de elementos lógicos criado para indicar os elementos que devem ser acessados foi criado manualmente, mas é muito comum que esse vetor seja criado a partir de comparações. Suponha que do vetor *numeros* tenha-se interesse em acessar somente os elementos com valores maiores que 300:

```
numeros
```

```
## [1] 290 320 270 302 329 307
```

```
numeros > 300
```

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

```
numeros[numeros > 300]
```

```
## [1] 320 302 329 307
```

que é equivalente a fazer manualmente

```
numeros[c(F,T,F,T,T,T)]
```

```
## [1] 320 302 329 307
```

O princípio da reciclagem também será utilizado na indexação utilizando variáveis lógicas. Pode-se, por exemplo utilizar-se dessa propriedade para selecionar apenas os elementos em índices pares fazendo:

```
numeros
```

```
## [1] 290 320 270 302 329 307
```

```
numeros[c(F,T)]
```

```
## [1] 320 302 307
```

a reciclagem ocorre de forma que o indicador dos índices *c(F,T)* na primeira posição é Falso e portanto o priemeiro elemento não será acessado, o segundo indicador é verdadeiro e portanto e segundo elemento é acessado. Para decidir se o terceiro elemento será acessado o vetor de indicador dos índices é reciclado de forma que para o terceiro elemento o indicador é falso e para o quarto é positivo e a reciclagem prossegue enquanto for necessário, dado o comprimento do vetor que armazena os valores.

Ordenação de vetores

A função *sort()* pode ser utilizada para ordenação de um vetor conforme o exemplo:

```
numeros
```

```
## [1] 290 320 270 302 329 307
```

```
sort(numeros)
```

```
## [1] 270 290 302 307 320 329
```

tendo como possibilidade a ordenação de forma decrescente por meio da opção vista abaixo:

```
numeros
```

```
## [1] 290 320 270 302 329 307
```

```
sort(numeros, decreasing = TRUE)
```

```
## [1] 329 320 307 302 290 270
```

Em algumas situações é importante saber as posições ordenadas dos dados. Suponha que tenha-se interesse em ordenar o vetor de **numeros** manualmente. Deveriamos primeiro pegar o elemento que está na posição 3 pois esse é o menor número, depois pegar o número que está na posição 1 que é o segundo menor número e as posições em que se deveria pegar os próximos números são 4, 6, 2, 5. E dessa forma ordenariamos o vetor:

```
numeros[c(3,1,4,6,2,5)]
```

```
## [1] 270 290 302 307 320 329
```

Esse processo que foi feito mentalmente, para encontrar as posições em que estão os valores de forma ordenada, é implementado pela função *order()*

```
order(numeros)
```

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

```
numeros[order(numeros)]
```

```
## [1] 270 290 302 307 320 329
```

tendo também a opção de buscar as posições ordenadas de forma decrescente

```
numeros[order(numeros, decreasing = TRUE)]
```

```
## [1] 329 320 307 302 290 270
```

A diferença entre as funções *sort* e *order* será importante quando deseja-se ordenar todo um conjunto de dados pelos valores de uma determinada variável de interesse.

Coerção

Os vetores são implementados em R como objetos ditos homogêneos, em que cada um de seus elementos deve ser do mesmo tipo. Um vetor numérico deve conter somente números, um vetor de caracteres deve conter somente elementos do tipo caractere. Para que a homogeneidade do vetor seja mantida elementos de tipos diferentes são transformados para que tenham o mesmo tipo por coerção. Veja nos exemplos a seguir o comportamento da linguagem R quando elementos de tipos diferentes são concatenados em um objeto homogêneo.

```
mistura <- c(1, "Ana", 3, "Beto")
mistura
```

```
## [1] "1"    "Ana"  "3"    "Beto"
```

note que aos elementos numéricos 1 e 3 foram adicionadas áspas, de forma que esses elementos foram transformados para o tipo caracter para que o vetor mantivesse a propriedade de homegeneidade. Não é possível realizar uma operação matemática com esses elementos transformados, uma vez que não são mais numéricos:

```
mistura[1]
```

```
## [1] "1"
```

```
mistura[1] + 1
```

```
## Error in mistura[1] + 1: non-numeric argument to binary operator
```

Ao tentar concatenar em um vetor elementos do tipo lógico com elementos do tipo caractere a transformação também acontece, transformando os elementos do tipo lógico em elementos do tipo caractere conforme pode-se ver no exemplo a seguir com as áspas adicionadas aos elementos lógicos

```
combinado <- c(TRUE, "verdade", FALSE, "mentira", T, F)
combinado
```

```
## [1] "TRUE"    "verdade" "FALSE"   "mentira" "TRUE"    "FALSE"
```

Na concatenação de elementos do tipo lógico com elementos do tipo numérico os elementos do tipo lógico são convertidos em numéricos de forma que valores TRUE são transformados no valor numérico 1 e valores FALSE são transformados em valores numéricos 0.

```
reunido <- c(10, TRUE, 1000, FALSE, 200, T, 500, F)
reunido
```

```
## [1]   10    1 1000    0  200    1  500    0
```

Essas transformações entre tipos de elementos também podem ser feitas por meio de funções. As funções deste tipo tem seus nomes definidos pelo tipo de variável que se deseja obeter com a transformação.

Para transformar elementos em numérico utiliza-se **as.numeric()**

```
logicos <- c(T, F, TRUE, FALSE)
as.numeric(logicos)
```

```
## [1] 1 0 1 0
```

Para transformar elementos em lógicos utiliza-se **as.logical()**

```
numericos <- c(0, 5, 0, 5000)
as.logical(numericos)
```

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

e nesse tipo de transformação o valor zero é transformado em **FALSE** e os valores diferentes de zero em **TRUE**.

Para transformar elementos em caracteres utiliza-se **as.character()**

```
numericos
```

```
## [1]    0    5    0 5000
```

```
as.character(numericos)
```

```
## [1] "0"    "5"    "0"    "5000"
```

A partir do conhecimento dos objetos do tipo vetor uma extensão simples é para os objetos do tipo matriz que tem suas propriedades ilustradas na seção seguinte.

Relação com algumas das funções vistas nos exemplos desta seção

- c()
- operador :
- seq()
- rep()
- rnorm()
- rpois()
- sort()
- order()