- Tutorial
- Exercícios
- Apostila

9. Noções de Programação

R: Um Ambiente Orientado a Objetos

Atributos

Até esse ponto do curso, foi visto que existem no R funções, variáveis e vetores. Todos esses ítens são chamados genericamente de **objetos**.

Veremos no decorrer do curso vários outros **objetos** do R. A importância do conceito de **objeto** num ambiente de trabalho de análise de dados é que os objetos possuem **atributos**, os quais podem variar em função do tipo de objeto.

Vejamos um exemplo.

```
> zoo
 onça  anta  tatu guará
    4    10     2    45
> class( zoo )
[1] "numeric"
> length( zoo )
[1] 4
> names( zoo )
[1] "onça"  "anta"  "tatu"  "guará"
>
```

O vetor 'zoo' é um vetor de classe 'numeric', de comprimento ('length') 4 e com nomes ('names'): onça, anta, tatu e guará. Classe, comprimento e nomes são os atributos típicos de vetores.

Qualquer vetor sempre terá uma classe e um comprimento, mas o atributo 'names' é opcional:

```
> b
[1] 1 2 3 4 5 6 7 8
> class( b )
[1] "integer"
> length( b )
[1] 8
> names( b )
NULL
>
```

A função 'attributes' nos mostra os atributos de um objeto, mas é de uso limitado no caso de vetores:

```
> zoo
 onça  anta  tatu guará
    4    10     2    45
> attributes( zoo )
$names
[1] "onça"  "anta"  "tatu"  "guará"

> b
[1] 1 2 3 4 5 6 7 8
> attributes( b )
NULL
>
```

Funções

As funções do R também são objetos, mas da classe '`function`':

```
> class( ls )
[1] "function"
> class( log )
[1] "function"
> class( sin )
[1] "function"
>
```

No caso das funções, podemos associar a elas os **argumentos** que elas necessitam para serem executadas:

```
> args( ls )
function (name, pos = -1, envir = as.environment(pos), all.names = FALSE,
    pattern)
NULL
> args( log )
function (x, base = exp(1))
NULL
>
```

Algumas funções matemáticas, no entanto, tem sempre apenas um argumento e são consideradas **funções primitivas**:

```
> args( sin )
NULL
> sin
.Primitive("sin")
>
> args( exp )
NULL
> exp
.Primitive("exp")
```

```
>
```

Mundo dos Objetos

Um aspecto importante num ambiente orientado a objetos é que **tudo** o que o ambiente trabalha são objetos e o ambiente não pode trabalhar com nada que não seja um objeto conhecido. Inclui nessa categoria tudo aquilo que o R apresenta na tela, por isso toda saída do R pode ser guardada num objeto:

```
> length( zoo )
[1] 4
> zoo.comp = length( zoo )
> zoo.comp
[1] 4
> class( zoo )
[1] "numeric"
> zoo.class = class( zoo )
> zoo.class
[1] "numeric"
> class( zoo.class )
[1] "character"
> names( zoo )
[1] "onça"  "anta"  "tatu"  "guará"
> class( names( zoo ) )
[1] "character"
> length( names( zoo ) )
[1] 4
>
```

Quando o R nos mostra, como resultado de uma operação, valores como `'NULL'` e `'integer(0)'` ele está dizendo que o resultado é **vazio**, isto é, não há resultado:

```
> b
[1] 1 2 3 4 5 6 7 8
> names( b )
NULL
> b[ b > 10 ]
integer(0)
>
```

Veja que o valor `'NULL'` é um valor válidos que podem ser utilizados.

```
> zoo2 = zoo
> zoo2
 onça  anta  tatu guará
    4    10     2    45
> names( zoo2 )
[1] "onça"  "anta"  "tatu"  "guará"
> names( zoo2 ) = NULL
```

```
> zoo2
[1]  4 10  2 45
> names( zoo2 )
NULL
>
```

Exercícios

Exercício 7.1: *Freqüência de Espécies*

Considere o vetor com nome de espécies:

```
  sp
```

[1] “Myrcia sulfiflora” “Syagrus romanzoffianus” “Tabebuia cassinoides” [4] “Myrcia sulfiflora”

Para obter a freqüência das espécies podemos usar a função 'table':

```
> table( sp )
sp
     Myrcia sulfiflora Syagrus romanzoffianus   Tabebuia cassinoides
                     2                       1                      1
>
```

Qual a classe do objeto que a função 'table' retorna? Quais são os seus attributos?

Exercício 7.2: *Classe da Classe*

Qual a classe do objeto produzido pelo comando 'class(x)'?

Construindo Funções Simples

A Estrutura Básica de uma Função

Toda manipulação de dados e análises gráficas e estatísticas no R são realizadas através de funções. Entretanto, você não precisa ser um programador experimentado para construir algumas funções simples para facilitar a atividade de manipulação de dados.

A estrutura básica de uma função é:

```
> minha.funcao <- function( argumento1, argumento2, argumento3, . . .)
                                    {

                                                comando 1

                                                comando 2
                                                comando 3
```

```
                                        . . .
                                        comando n
                                        return("resultado")
                                }
```

Os elementos dessa expressão são:

- **minha.funcao** é o nome que a nova função receberá;
- **function** é a expressão no R que cria uma nova função;
- **entre as chaves "{}"** são listados os comandos da função, sempre com um comando por linha;
- **entre os parênteses "()"** são listados (separados por vírgula) os argumentos necessários a função;
- **comando return("resultado")** retorna os resultados, caso falte, será apresentado o resultado do último comando (comando n).

Vejamos alguns exemplos simples:

```
##criar um vetor de dados com 20 valores aleatórios de uma distribuição
Poisson

dados.dens<-rpois(20,lambda=6)

##funcão para calcular média

media.curso <-function(x,rmNA=TRUE)
    {
    soma=sum(x)
    nobs=length(x)
    media=soma/nobs
    return(media)
    }

##Vamos agora preparar uma função mais elaborada, considerando a
##presença e excluíndo NA por padrão, e lançando mensagem na tela
##sobre o número de NAs removidos. Note que é uma função com dois argumentos
##que permite ao usuário tomar a decisão de remover ou não NAs e avisando,
##diferente da função mean()


media.curso <-function(x,rmNA=TRUE)
    {
    if(rmNA==TRUE)
        {
        dados=(na.omit(x))
        dif=length(x)-length(dados)
        cat("\t", dif," valores NA excluídos\n")
        }
        else
        {
        dados=x
```

```
        }
    soma=sum(dados)
    nobs=length(dados)
    media=soma/(nobs)
    return(media)
    }

###calcular a média do objeto dados
media.curso(dados.dens)


##########################################
###função para calcular variância

var.curso<-function(x)
    {
    media=media.curso(x)
    dados=na.omit(x)
    disvquad=(dados-media)^2
    variancia=sum(disvquad)/(length(dados)-1)
    return(variancia)
    }

###Calcular a variância de dados
var.curso(dados.dens)

###Tomando dados.dens como a contagem de uma espécie em uma amostra de 20
parcelas de 20x20m,
###podemos verificar o padrão de dispersão dessa espécie, utilizando o
Índice de Dispersão (razão variância / média)

ID.curso<-function(x)
    {
    id=var.curso(x)/media.curso(x)
    return(id)
    }

##Calcular o  coeficiente de dispersão

ID.curso(dados.dens)

## quando o valor é próximo a 1 a distribuição é considerada aleatória.
## podemos fazer um teste de significância  pela aproximação com o valor
Qui-Quadrado
para verificar a significância dos dados

test.ID <- function(x)
        {
         dados=na.omit(x)
       med=media.curso(x)
       dev.quad=(dados-med)^2
```

```
        qui=sum(dev.quad)/med
          critico.qui<-qchisq(c(0.025,0.975),df=(length(dados)-1))
             if(critico.qui[1]<=qui & critico.qui[2]>=qui)
             { cat("\t distribuição aleatória para alfa=0.05\n")}
           else{}
          if(qui < critico.qui[1])
                { cat("\t","distribuição agregada, p<0.025 \n")}
            else{}
         if(qui>critico.qui[2])
         { cat("\t","distribuição regular, p>0.975 \n")}
        resulta=c(qui,critico.qui)
        names(resulta)<-c("qui-quadrado", "critico 0.025", "critico 0.975")
          return(resulta)
          }

############
```

Exercícios

Exercício 7.3: *QUE FRIO!*

Construa uma função que calcula automaticamente o valor de graus Celsius, sabendo-se a temperatura em Fahrenheit.

C° = 5/9 * (F°(temperatura dada) - 32)

Exercício 7.4: *Somatório do Primeiros Números Naturais*

Construa uma função que calcula o somatório dos primeiros *n* números naturais.

Por exemplo se *n=4* a função deve retornar o valor: *1+2+3+4*.

Exercício 7.5: *Índices de Dispersão I*

Existe uma série de índices de dispersão baseados em dados de contagem para verificar o padrão espacial de uma espécie.

Alguns deles são:

- **Razão Variância-Média**: ID = variância / média;
- **Coeficiente de Green**: IG = (ID-1)/(n-1);
- **Índice de Morisita**:

$$I_\delta = n \frac{\left(\sum_{i=1}^{n} x_i^2 - \sum_{i=1}^{n} x_i\right)}{\left(\sum_{i=1}^{n} x_i\right)^2 - \sum_{i=1}^{n} x_i}$$

onde:

n = tamanho da amostra; xi= número de indivíduos na i -ésima unidade amostral

Construa uma função para cada um desses índices, assumindo como argumento os valores de xi. Aplique aos dados de caixetais, verificando a dispersão da árvores de caixeta em cada caixetal.

Definindo Argumentos

Todos argumentos de uma função tem seu respectivo nome. Ao evocar a função podemos fazê-lo de duas formas:

- utilizando o **nome** dos argumentos **em qualquer ordem**;
- utilizando a **ordem** dos argumentos, mas **omitindo** os nomes.

```
> plot( col="red", pch=2, y=egr$ht, x=egr$dap )
> plot( egr$dap, egr$ht )
```

Para qualquer argumento podemos definir um **valor default** apresentando esse valor junto com argumento na definição da função:

```
> myplot <- function(..., col="red") { plot(..., col="red") }
> myplot( cax$dap, cax$h )
> myplot( ht ~  dap, data=egr )
```

O exemplo acima também mostra a função do argumento **"..."**. Esse argumento representa **qualquer argumento adicional** que desconhecemos, mas que desejamos que seja passado para as funções dentro da função que estamos construindo.

Exercícios

Exercício 7.6: *Gráfico de Whittaker*

Faça uma função para construir o gráfico de diversidade de espécies de Whittaker: logaritmo da abundância contra a ordem (descrescente) da abundância das espécies. Construa essa função de forma que qualquer parâmetro gráfico possa ser alterado.

Trabalhando com Funções mais Complexas

Um Aspecto Prático

Para saber qual é o editor padrão do R use o comando:

```
> getOption("editor")
[1] "vi"
>
```

Para alterar o editor padrão use o comando:

```
> options( editor= "gedit" )       # Faz o editor "gedit" ser o editor padrão do R
```

No caso de editar sua função num editor externo ao R (p.ex., no arquivo 'minhas-funcoes.R'), você traz o código para dentro do R utilizando o comando **“source”**:

```
> source( "minhas-funcoes.R" )
```

É importante que o arquivo editado externamente ('minhas-funcoes.R') seja um arquivo **ASCII** sem qualquer símbolo especial.

Exercícios

Exercícios: *Editando Funções Externamente*

Experimente definir um editor com o qual você consiga trabalhar ('gedit'?) e refaça os exercícios anteriores salvando todos os códigos num arquivo externo.

Exercícios: *Índices de Diversidade de Espécies*

Construa funções para computar os seguintes índices de diversidade de espécies:

- Índice de Shannon: H = - ∑ (p*i* * ln(p*i*))
- Índice de Simpson: D = ∑(p*i*^2)

onde p*i* é a proporção da espécie *i*

Considere que o argumento de sua função será uma matriz com a abundância das espécies sendo as parcelas amostradas nas colunas. Considere a possibilidade de haver NA nessa matrix e a remoção dele.

Realizando "Loops"

Em linguagem de programação um **loop** é quando você força um programa a executar uma série de comandos repedidas vêzes.

A estrutura de loop no R é:

```
for( "variável"  in  "vetor de valores")
{
       comando 1
       comando 2
       comando 3
       . . .
       comando n
}
```

A palavra **for** é o chamado do loop. Dentro dos parênteses se define uma variável seguida da palavra **in** e um vetor de valores que a variável deverá assumir. Dentro das chaves se lista os comandos que

devem ser repeditos a cada passo do loop.

Vejamos um exemplo: *Convergência da distribuição t de Student para distribuição Normal Padronizada*:

```
> #
> # Convergência da distribuição t de Student para distribuição Normal
Padronizada
> #
> curve(dnorm(x), from=-4, to=4, col="red", lwd=6)
> for(gl in 1:200)
+ {
+      curve(dt(x, gl), -4, 4, add=TRUE, col="green")
+ }
>
```

No exemplo acima temos:

- 'gl' é a variável definida para o loop;
- '1:200' é o vetor de valores que a variável assumirá, logo, o loop será repetido 200 vêzes.

Exercícios

Exercícios: *Loop para Demonstrar o TCL*

Construa uma função para demonstrar o Teorema Central do Limite.

Solução Vetorial x Loop

Sendo um **ambiente vetorial**, os *loops* não são uma opção muito eficiente para computação dentro do R. Em geral, o R é mais eficiente se encontrarmos uma **solução vetorial** para problemas de computação que aparentemente exigem um loop. A solução vetorial, entretanto, costuma ser mais exigente em termos do tamanho de memória RAM do computador.

Considere o problema o seguinte problema: temos a localização espacial de plantas num plano cartesiano com coordenadas **(x,y)**. Por exemplo:

```
> x = runif(100)
> y = runif(100)
> plot(x,y)
```

O objetivo é obter as **distâncias** entre as plantas duas-a-duas. Primeiro consideremos uma solução através de loop:

```
inter.edist = function(x, y)
{
    n = length(x)
    dist  <- c()
    for(i in 1:(n-1))
```

```
    {
        for(j in (i+1):n)
        {
            dist  <- c(dist, sqrt( (x[i] - x[j])^2 + (y[i] - y[j])^2 ))
        }
    }
    dist
}
```

Consideremos agora uma solução vetorial:

```
inter.edist.v = function(x, y)
{
    xd <- outer( x, x, "-" )
    yd <- outer( y, y, "-" )
    z <- sqrt( xd^2 + yd^2 )
    dist <- z[ row(z) > col(z) ]
    dist
}
```

Qual dessas soluções é mais eficiente em termos do uso do tempo?

```
> x = runif(100)
> y = runif(100)
>
> system.time( inter.edist( x, y ) )
[1] 0.140 0.008 0.149 0.000 0.000
>
> system.time( inter.edist.v( x, y ) )
[1] 0.008 0.000 0.009 0.000 0.000
```

Não tente rodar o exemplo acima com 1000 ou mais observações, pois o tempo fica **realmente longo** para versão em loop.

CONCLUSÃO: use apenas **pequenos loops** no R!!!

Exercícios

Exercícios: *Tabela de Fitossociologia*

Construa uma função que gera uma tabela de fitossociologia. Utilize os dados de caixeta (Conjunto de Dados: Levantamento em Caixetais) como teste.

APÊNDICE: Tabela de Operadores do R

Outro aspecto formal importante da linguagem R é a ordem de prioridade de seus operadores. Além das regras de precedência usuais para as operações matemáticas, é essencial conhecer a prioridade dos outros operadores:

```
=============================================================
======
 OPERADOR          DESCRIÇÃO
PRIORIDADE
=============================================================
======
  $                seleção de componentes
ALTA
 [  [[             indexação                                 |
 ^                 potência                                  |
 -                 menos unitário                            |
 :                 operador de seqüência                     |
 %nome%            operadores especiais                      |
 *  /              multiplicação, divisão                    |
< > <= >= == !=    comparação                                |
 !                 não                                       |
 & &&  |  ||       e, ou                                     |
 ~                 fórmula estatística
\/
 <<- <- -> =       atribuição
Baixa
=============================================================
======
```