MAC0113 - Introdução à Computação para Ciências Humanas

Aula 15 - 1

Sejam bem-vindas, sejam bem-vindos!

Entre no link https://app.sli.do/event/duagqp6o ou e responda a primeira pergunta da aula.

R. Hirata Jr.

MAC0113 - Introdução à Computação para Ciências Humanas

Aula 15 - 2

Sejam bem-vindas, sejam bem-vindos!

Entre no link https://app.sli.do/event/fufsievy ou e responda a primeira pergunta da aula.

R. Hirata Jr.

Objetivos de hoje

- Ao final da aula de hoje você deve saber:
  - como os caracteres são representados
  - matrizes
  - leitura de um arquivo

Faça todos os exercícios abaixo

1. Dado um número inteiro positivo, que representa um ano, verifique se ele é um ano bissexto. Se for, imprima a mensagem: “O ano digitado é bissexto”. Se não for, imprima “O ano digitado não é bissexto”.
2. Dado um vetor com uma sequência de números positivos quaisquer, determine se a sequência está em ordem crescente.
3. Dado um vetor com uma sequência com as notas finais de MAC113, determinar quantos estudantes:
   - foram aprovados: nota final >= 5;
   - estão de recuperação: 3 <= nota final < 5;
   - foram reprovados: nota final < 3.

Um ano é bissexto se ele é múltiplo de 4 e não é múltiplo de 100, ou se ele é múltiplo de 400. Por exemplo, os anos 1700, 1800, 1900 não são anos bissextos. Os anos 1600, 2000 e 2020 o são.

Indicador de passagem

```
# inicializa o indicador como FALSE (TRUE)

indicador <- FALSE
while (<condição>) {
    # comandos
    if (<condição>) {
        indicador <- TRUE  # passou por aqui
        # comandos
    }
}
if (indicador==TRUE) {
       # comandos
}
```

Exercício 2

```
# Dado um vetor V com uma sequência de números positivos quaisquer, determine se a
# sequência está em ordem crescente.
i = 1
tamanhoV <- length(V)
ehCrescente <- TRUE # indicador de passagem
while ((ehCrescente==TRUE)&(i < tamanhoV)) {
    ehCrescente <- (V[i] < V[i+1])
    i = i + 1
}
if (ehCrescente==TRUE) {
    print("A sequência digitada é crescente.")
} else {
    print("A sequência digitada não é crescente.")
}
```

Exercício 3

```
# Dado um vetor V com uma sequência com as notas finais de MAC113, determinar quantos
# estudantes foram aprovados: nota final >= 5;  estão de recuperação: 3 <= nota final < 5; foram reprovados: nota final < 3.
noAprovados   <- 0
noRecuperacao <- 0
noReprovados  <- 0
i <- 1
tamanhoV <- length(V)
while (i <= tamanhoV) {
     if (V[i] >= 5.0) {
          noAprovados <- noAprovados + 1
     } else if (V[i] >= 3.0) {
          noRecuperacao <- noRecuperacao + 1
     } else {
          noReprovados <- noReprovados + 1
     }
     i <- i+1
}
cat("Foram aprovados ",noAprovados," estudantes.\n")
cat("Foram reprovados ",noReprovados," estudantes.\n")
cat("Estão de recuperação ",noRecuperacao," estudantes.\n")
```

ASCii e UTF-8

Existem várias codificações (*enconding*) de texto. As mais comuns são

- *ASCii*: que representa apenas letras do alfabeto latino sem acento, números e pontuações usando no máximo 8 bits

  https://en.wikipedia.org/wiki/ASCII

- *UTF-8*: que representa letras do alfabeto latino e de outros alfabetos, incluindo acentos e diacríticos (p.ex., ç), números, pontuações e outros símbolos (flechas, operadores matemáticos etc), usando uma quantidade de bits variável.

  https://en.wikipedia.org/wiki/UTF-8

O padrão nas versões recentes do R é usar UTF-8.

Codificação de caracteres

Os caracteres no computador são representados por números inteiros positivos. A codificação (encoding) de um arquivo é a lista de códigos usados para representar caracteres. A função utf8ToInt devolve o código de um caractere.

```
V <- c("A","B","C","D","E","F","R","S","T","U","V","X","Y","Z")
i <- 1
tamanhoV <- length(V)
while (i <= tamanhoV) {
  cat(utf8ToInt(V[i])," ")
  i <- i+1
}
cat("\n")
```

Codificação de caracteres

Os caracteres no computador são representados por números inteiros positivos. A codificação (encoding) de um arquivo é a lista de códigos usados para representar caracteres. A função utf8ToInt devolve o código de um caractere.

```
V <- c("a","b","c","d","e","f","r","s","t","u","v","x","y","z")
i <- 1
tamanhoV <- length(V)
while (i <= tamanhoV) {
  cat(utf8ToInt(V[i])," ")
  i <- i+1
}
cat("\n")
```

Codificação de caracteres

Os caracteres no computador são representados por números inteiros positivos. A codificação (encoding) de um arquivo é a lista de códigos usados para representar caracteres. A função utf8ToInt devolve o código de um caractere.

```
V <- c("a","b","c","d","e","f","r","s","t","u","v","x","y","z")


for (letra in V) {
  cat(utf8ToInt(letra)," ")

}
cat("\n")
```

Codificação de caracteres

Os caracteres no computador são representados por números inteiros positivos. A codificação (encoding) de um arquivo é a lista de códigos usados para representar caracteres. A função utf8ToInt devolve o código de um caractere.

```
V <- c("á","à","ã","â","ä","å")


for (letra in V) {
  cat(utf8ToInt(letra)," ")

}
cat("\n")
```

Matrizes

Uma matriz é um arranjo multidimensional de objetos do mesmo tipo

Por exemplo, uma imagem pode ser representada por uma matriz de valores inteiros:

0	0	1	9
3	1	2	7
1	1	4	3
3	3	3	5
0	2	1	1

Matrizes

Existem várias maneiras de se criar uma matriz. Uma delas é usando o operador c
Por exemplo, para a matriz abaixo, poderíamos fazer:

```
> V = c(0,3,1,3,0,0,1,1,3,2,1,2,4,3,1,9,7,3,5,1)
> dim(V) = c(5,4)
dim(V)
[1] 5 4
> V
     [,1] [,2] [,3] [,4]
[1,]    0    0    1    9
[2,]    3    1    2    7
[3,]    1    1    4    3
[4,]    3    3    3    5
[5,]    0    2    1    1
```

0	0	1	9
3	1	2	7
1	1	4	3
3	3	3	5
0	2	1	1

Matrizes

Uma outra maneira é usando o operador cbind (c de column)
Por exemplo, para a matriz abaixo, poderíamos fazer:

```
> W <-  cbind(c(0,3,1,3,0),c(0,1,1,3,2),c(1,2,4,3,1),c(9,7,3,5,1))
> W
     [,1] [,2] [,3] [,4]
[1,]    0    0    1    9
[2,]    3    1    2    7
[3,]    1    1    4    3
[4,]    3    3    3    5
[5,]    0    2    1    1
```

0	0	1	9
3	1	2	7
1	1	4	3
3	3	3	5
0	2	1	1

Matrizes

Uma outra maneira é usando o operador rbind (r de row)
Por exemplo, para a matriz abaixo, poderíamos fazer:

```
> W2 <-  rbind(c(0,0,1,9),c(3,1,2,7),c(1,1,4,3),c(3,3,3,5),c(0,2,1,1))
> W2
     [,1] [,2] [,3] [,4]
[1,]    0    0    1    9
[2,]    3    1    2    7
[3,]    1    1    4    3
[4,]    3    3    3    5
[5,]    0    2    1    1
```

0	0	1	9
3	1	2	7
1	1	4	3
3	3	3	5
0	2	1	1

Matrizes

Uma outra maneira é usando o operador matrix, com parâmetros ncol e nrow
Por exemplo, para a matriz abaixo, poderíamos fazer:

```
> M <-  matrix(c(0,3,1,3,0,0,1,1,3,2,1,2,4,3,1,9,7,3,5,1),ncol=4,nrow=5)
> M
     [,1] [,2] [,3] [,4]
[1,]    0    0    1    9
[2,]    3    1    2    7
[3,]    1    1    4    3
[4,]    3    3    3    5
[5,]    0    2    1    1
```

0	0	1	9
3	1	2	7
1	1	4	3
3	3	3	5
0	2	1	1

Matrizes

Uma vez criada, podemos acessar, ou acessar e modificar os valores da matriz usando um, ou mais índices. Por exemplo:

```
> M[1,1]
[1] 0
> M[1,2]
[1] 0
> M[2,1]
[1] 3
> M[5,4]
[1] 1
> M[2,2]
[1] 1
> M[4,3]
[1] 3
```

```
> M
     [,1] [,2] [,3] [,4]
[1,]    0    0    1    9
[2,]    3    1    2    7
[3,]    1    1    4    3
[4,]    3    3    3    5
[5,]    0    2    1    1
```

Matrizes

Uma vez criada, podemos acessar, ou acessar e modificar os valores da matriz usando dois, ou mais índices (casos com mais de 2 dimensões). Por exemplo:

```
> M[3,2] <- 50
> M
     [,1] [,2] [,3] [,4]
[1,]    0    0    1    9
[2,]    3    1    2    7
[3,]    1   50    4    3
[4,]    3    3    3    5
[5,]    0    2    1    1
```

Antes

```
> M
     [,1] [,2] [,3] [,4]
[1,]    0    0    1    9
[2,]    3    1    2    7
[3,]    1    1    4    3
[4,]    3    3    3    5
[5,]    0    2    1    1
```

Matrizes

Podemos acessar linhas, ou colunas inteiras, usando o fatiamento. Por exemplo:

```
# linha 3
> W[3,]
[1] 1 50 4 3

# coluna 2
> W[,2]
[1]  0  1 50  3  2
```

```
> W
     [,1] [,2] [,3] [,4]
[1,]    0    0    1    9
[2,]    3    1    2    7
[3,]    1   50    4    3
[4,]    3    3    3    5
[5,]    0    2    1    1
```

Matrizes

Podemos acessar mais de uma linha, ou coluna inteiras, usando o fatiamento e o operador c, também. Por exemplo:

```
# linhas 1, 3 e 5
> W[c(1,3,5),]
    [,1] [,2] [,3] [,4]
[1,]    0    0    1    9
[2,]    1   50    4    3
[3,]    0    2    1    1
```

```
> W
     [,1] [,2] [,3] [,4]
[1,]    0    0    1    9
[2,]    3    1    2    7
[3,]    1   50    4    3
[4,]    3    3    3    5
[5,]    0    2    1    1
```

Matrizes

Podemos acessar mais de uma linha, ou coluna inteiras, usando o fatiamento e o operador c, também. Por exemplo:

```
# colunas 2 e 4
> W[,c(2,4)]
     [,1] [,2]
[1,]    0    9
[2,]    1    7
[3,]   50    3
[4,]    3    5
[5,]    2    1
```

```
> W
     [,1] [,2] [,3] [,4]
[1,]    0    0    1    9
[2,]    3    1    2    7
[3,]    1   50    4    3
[4,]    3    3    3    5
[5,]    0    2    1    1
```

Leitura de arquivo

Existem várias maneiras de armazenar dados e, depois, carregá-los (ler) em R.

O assunto arquivos é muito extenso e cada vez menos importante, graças à disponibilidade dos dados de forma aberta e acessíveis por pacotes do R. Por exemplo, o pacote quantmod (**Quantitative Financial Modelling Framework**) que pode ser encontrado aqui:

https://cran.r-project.org/web/packages/quantmod/index.html

Para instalar o pacote no seu R, faça

```
install.packages("quantmod")
```

Leitura de arquivo

Existem centenas de pacotes para o R. Olhe aqui:

https://cran.r-project.org/web/packages/

Uma vez que o pacote está instalado, para usá-lo, você deve carregar com o comando library:

```
library("quantmod")
```

Leitura de arquivo

Agora, você pode se divertir. Com o comando getSymbols, você vai criar um dataframe chamado PETR3.SA:

```
getSymbols("PETR3.SA", src = "yahoo", from = "2021-05-01", to = "2021-05-29")
```

```
> head(PETR3.SA)
```

	PETR3.SA.Open	PETR3.SA.High	PETR3.SA.Low	PETR3.SA.Close	PETR3.SA.Volume	PETR3.SA.Adjusted
2021-05-03	23.17	23.26	22.77	23.01	15331500	23.01
2021-05-04	23.06	23.26	22.48	22.48	15023900	22.48
2021-05-05	22.76	23.57	22.68	23.40	21290700	23.40
2021-05-06	23.25	23.44	23.00	23.07	12919700	23.07
2021-05-07	23.22	23.93	23.04	23.93	23588010	23.93
2021-05-10	24.21	24.46	24.03	24.23	21003500	24.23

Leitura de arquivo

No entanto, algumas vezes é importante usar arquivos e, neste caso, há duas informações muito importantes que você precisa saber sobre eles:

- Nome, com o caminho completo
- Formato: csv (comma separated values)
  tsv (tab separated values)
  etc

Para carregar um arquivo no R, você pode usar as várias versões do read (read.csv, read.csv2, read.table). O que muda nesses comandos são os parâmetros default. Para ler uma tabela com o separador “;”, você pode fazer:

<nome do objeto> = read.csv(<nome do arquivo>,sep=”;”)

Se o separador for um tab, o separador será o “\t”.

Leitura de arquivo

Os comandos read carregam o arquivo e criam um dataframe (uma lista de listas), que é parecido com uma matriz.

Uma vez carregado e atribuído para um objeto do tipo dataframe, você pode começar a usá-lo. Por exemplo:

```
> dados <- read.csv("/home/hirata/MEGA/Disciplinas/MAC0113/EPs/MAC0113-EP1.csv",sep=";")
```

Usando o comando str, você pode ter uma ideia do objeto:

```
> str(dados)
'data.frame':    600 obs. of  3 variables:
$ Nome.do.objeto: chr  "ring light" "garrafa termica" "Lata de Nescau" "vaso" ...
$ circunferência: num  83 21.4 22.9 32.5 18.5 ...
$ diâmetro      : num  26 6.4 7.3 10.2 5.2 8.5 8.3 18 5.4 6 ...
```


INSTITUTO DE
MATEMÁTICA
E ESTATÍSTICA
IME
50
UNIVERSIDADE
DE SÃO PAULO

Obrigado!