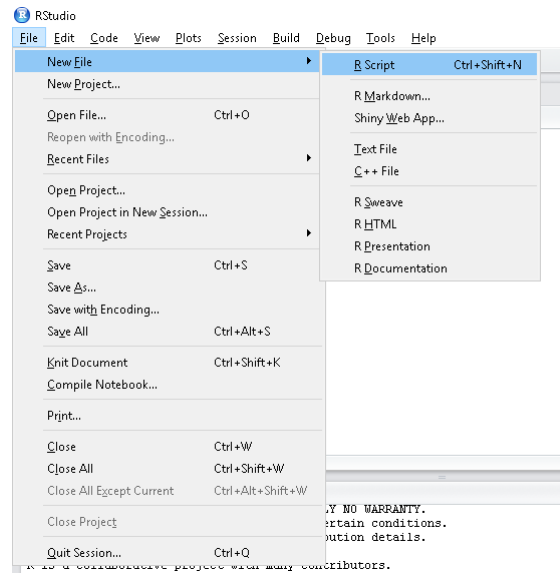


# Conceitos Básicos – Como Executar um Script

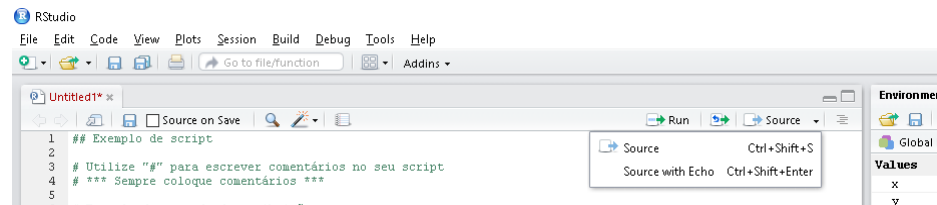
## Scripts

- Arquivos textos com lista de comandos para serem executados
- Para criar um novo script: [File] [R Script] ou *Ctrl+Shift+N*



## - Execução:

- Linha-a-Linha ou parcial: selecionar a linha + *Ctrl + Shift + Enter*
- Via menu:

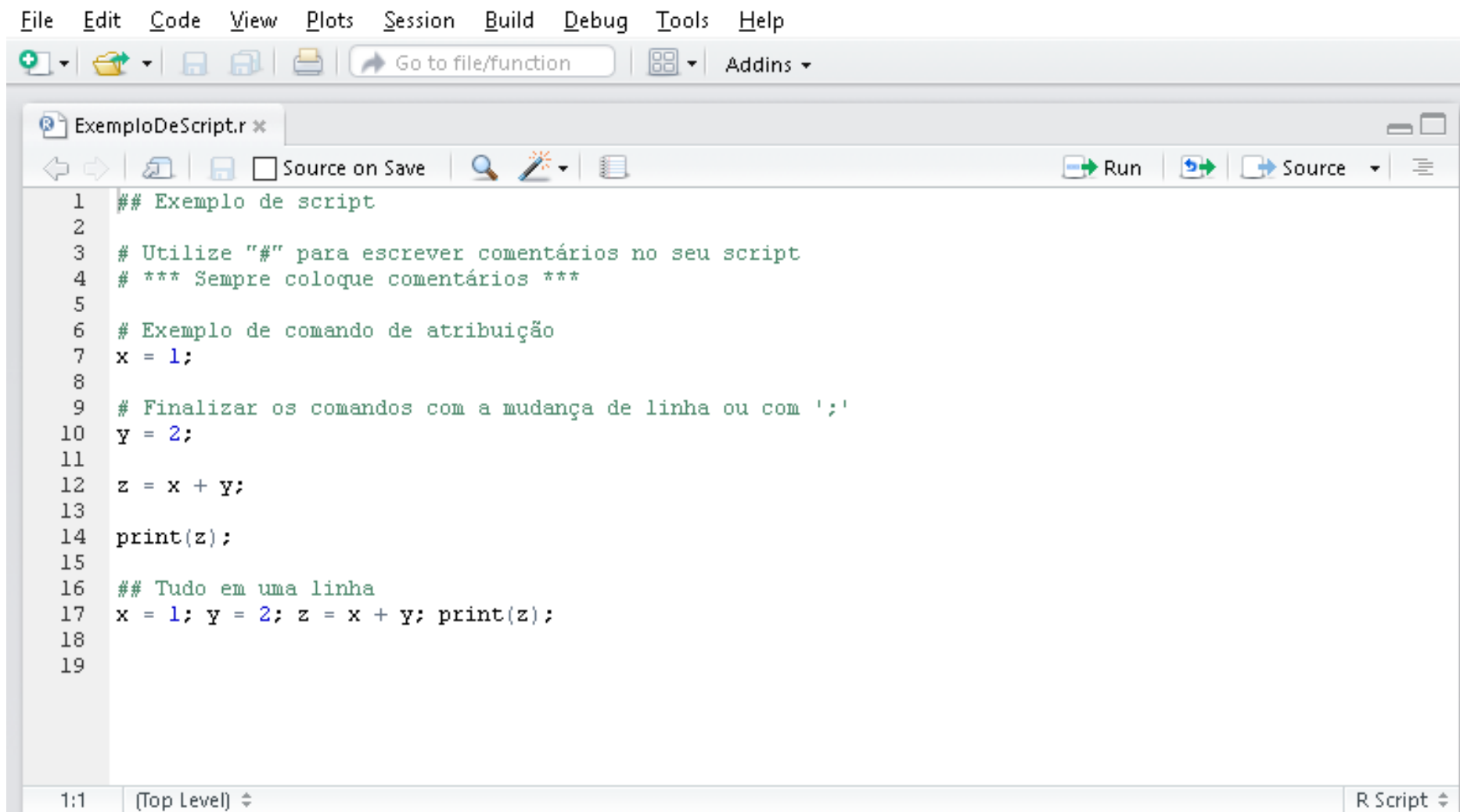


- Podem ser gravados para serem executados posteriormente

# Conceitos Básicos – Como Executar um Script

## Exemplo de Script:

C:/Public/FGV-MPFE/Materias/MPFE-Metodos Quantitativos/Modulos/Introdução ao R/R - RStudio



The screenshot shows the RStudio interface with the following elements:

- Menu Bar:** File, Edit, Code, View, Plots, Session, Build, Debug, Tools, Help.
- Toolbar:** Includes icons for file operations (new, open, save, print), a search bar labeled 'Go to file/function', and an 'Addins' dropdown.
- Script Editor:** Displays the file 'ExemploDeScript.r' with the following R code:
 

```
1 ## Exemplo de script
2
3 # Utilize "#" para escrever comentários no seu script
4 # *** Sempre coloque comentários ***
5
6 # Exemplo de comando de atribuição
7 x = 1;
8
9 # Finalizar os comandos com a mudança de linha ou com ';'
10 y = 2;
11
12 z = x + y;
13
14 print(z);
15
16 ## Tudo em uma linha
17 x = 1; y = 2; z = x + y; print(z);
18
19
```
- Editor Controls:** Includes navigation arrows, a 'Source on Save' checkbox, and buttons for 'Run' and 'Source'.
- Status Bar:** Shows '1:1' and '[Top Level]' on the left, and 'R Script' on the right.

# Conceitos Básicos – Como Executar um Script

Execução de script gravados:

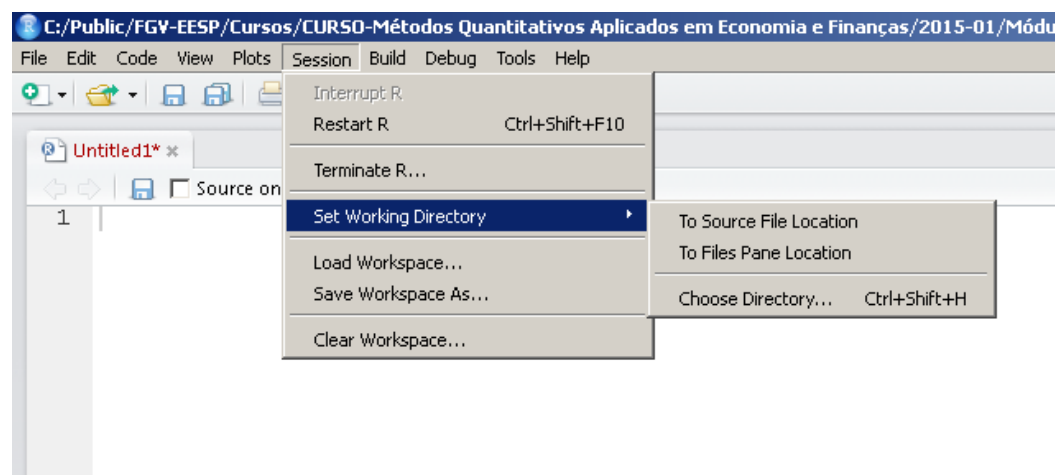
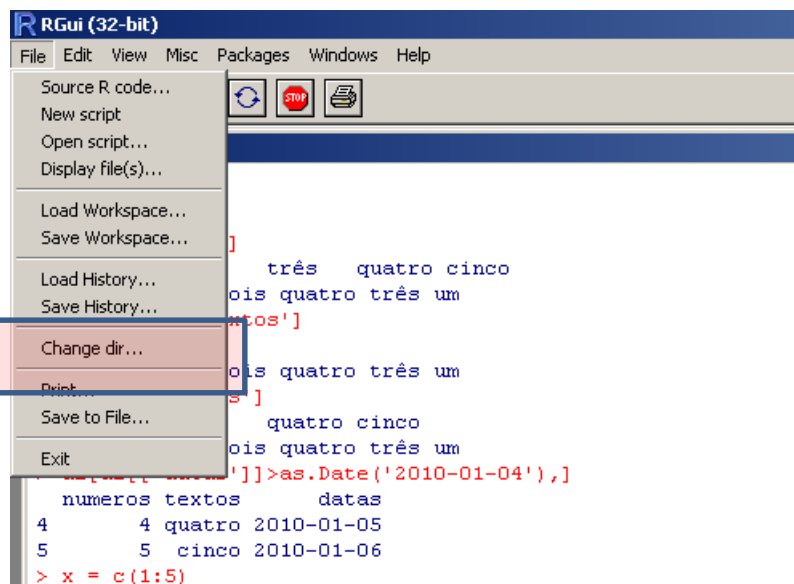
```
source (NomeDoScript) ;
```

```
source (CaminhoAbsolutoDoScript) ;
```

```
source (CaminhoRelativoDoScript) ;
```

**Atenção:**

- Verificar o diretório corrente para se referenciar e executar o script.
- No R para indicar diretório, utilizar “/” e não “\”. Ex: C:/tmp/exemplo.r



# Conceitos Básicos – Tipos de Dados

## Principais Tipos de Dados:

- Numeric

### Números Reais

```
> x = 10
> x = 20/3
> print(x)
[1] 6.666667
```

- String

### Sequencia de caracteres ou texto

Indicado utilizando aspas simples (‘ ’) ou aspas duplas (“ ”)

```
> umastring = "uma string"
> uma_string = "uma string"
> outra_string = 'outra string'
> print (uma_string)
[1] "uma string"
> print(outra_string)
[1] "outra string"
```

# Conceitos Básicos – Tipos de Dados

## Principais Tipos de Dados:

- Boolean

Verdadeiro/Falso

```
> x = TRUE
> y = FALSE
> print(x)
[1] TRUE
> print(y)
[1] FALSE
```

- Date

Padrão de representação: aaaa-mm-dd

Indicada como uma string entre aspas simples ou duplas

Utilizar a função `as.Date(<string>)`

```
> hoje = Sys.Date()
> print(hoje)
[1] "2015-01-07"
> ontem = as.Date("2015-01-06")
> print(ontem)
[1] "2015-01-06"
```

# Conceitos Básicos – Tipos de Dados

## Principais Tipos de Dados:

- NA

*Not Available:* A variável está definida mas o valor não está disponível

```
> x = NA
> is.na(x)
[1] TRUE
> y = 10
> is.na(y)
[1] FALSE
```

- NULL

Variável não está definida. Não utiliza Espaço de memória do computador

```
> x = NULL
> is.null(x)
[1] TRUE
> is.na(x)
logical(0)
Warning message:
In is.na(x) : is.na() applied to non-(list or vector) of type 'NULL'
> y=NA
> is.null(y)
[1] FALSE
```

# Conceitos Básicos – Operadores

## Operador de Atribuição

Operação	Símbolo	Exemplo	Resultado
Atribuição	= <-	x = 5 x <- 5	
Range	:	3 : 5	3 4 5

## Operadores Aritméticos

Operação	Símbolo	Exemplo
Adição	+	5 + 3
Subtração	-	5 - 3
Multiplicação	*	5 * 3
Divisão	/	5 / 3
Potenciação	^	5 ^ (1 / 2)

# Conceitos Básicos – Operadores

## Operadores Relacionais

Operador	Símbolo	Exemplo	Resultado
Igualdade	<b>==</b>	5 == 3	FALSE
Diferença	<b>!=</b>	5 != 3	TRUE
Maior que	<b>&gt;</b>	5 > 3	TRUE
Menor que	<b>&lt;</b>	5 < 3	FALSE
Maior ou Igual	<b>&gt;=</b>	5 >= 3	TRUE
Menor ou Igual	<b>&lt;=</b>	5 <= 3	FALSE

## Operadores Lógicos

Operador	Símbolo	Exemplo	Resultado
E vetorial	<b>&amp;</b>	(TRUE, FALSE) & (TRUE, TRUE)	(TRUE, FALSE)
OU vetorial	<b> </b>	(TRUE, FALSE)   (TRUE, TRUE)	(TRUE, TRUE)
E escalar	<b>&amp;&amp;</b>	TRUE && FALSE	FALSE
OU escalar	<b>  </b>	TRUE    FALSE	TRUE
Negação	<b>!</b>	!(5 < 3)	TRUE



# Conceitos Básicos – Estruturas de Dados

- Vetores
- Todos os dados armazenados no R são vetores
- Estrutura unidimensional
- Para criar um vetor utilizar o comando “concatenar”: `c()`

```
> x = c(10,20,30)
> print(x)
[1] 10 20 30

> s = c('string 01',"string 02", "string 03")
> print(s)
[1] "string 01" "string 02" "string 03"

> b = c(TRUE,FALSE,TRUE)
> print(b)
[1] TRUE FALSE TRUE

> d = c(Sys.Date(),as.Date('2012-01-01'), as.Date('2013-10-01'))
> print(d)
[1] "2015-01-07" "2012-01-01" "2013-10-01"
```

# Conceitos Básicos – Estruturas de Dados

- Vetores (continuação)
- **Para acessar um elemento do vetor:** `NomeVariavel[<index>]`
- **Primeiro elemento é o 1**
- **Utilizar o operador Range ‘:’**

```
> x = c(10,20,30)
> x[2]
[1] 20
> s = c('string 01',"string 02", "string 03")
> s[2]
[1] "string 02"
> s[2:3]
[1] "string 02" "string 03"
> y = 1:10
> y[5]
[1] 5
> y[6:9]
[1] 6 7 8 9
```

# Conceitos Básicos – Estruturas de Dados

- Vetores (continuação)
- **Os elementos dos vetores podem ser nomeados e acessados pelo nome**

```
> n = c('primeiro'=1, 'segundo'=2, terceiro=3)
```

```
> n
```

```
primeiro  segundo terceiro
         1         2         3
```

```
> n[c('primeiro','terceiro')]
```

```
primeiro terceiro
         1         3
```

```
> n[c(1,3)]
```

```
primeiro terceiro
         1         3
```

```
> names(n)
```

```
[1] "primeiro" "segundo"  "terceiro"
```

```
> names(n) = c("um","dois","tres")
```

```
> n
```

```
um dois tres
  1    2    3
```

```
> n[c("um","tres")]
```

```
um tres
  1    3
```

# Conceitos Básicos – Estruturas de Dados

## Operações lógicas com vetores:

NomeVariavel[Expressão Lógica]

**Função** length: length(<VetorBase>)

**Operador** %in%: <VetorBase> %in% <VetorReferência>

```
> x = 1:20
```

```
> x
```

```
[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

```
> length(x)
```

```
[1] 20
```

```
> x[x>5 & x <10]
```

```
[1] 6 7 8 9
```

```
> x[x%%2==0]
```

```
[1] 2 4 6 8 10 12 14 16 18 20
```

```
> y = seq(0,20,by=2)
```

```
> y
```

```
[1] 0 2 4 6 8 10 12 14 16 18 20
```

```
> y %in% x
```

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

# Conceitos Básicos – Estruturas de Dados

- Arrays
- Estruturas de dados com uma, duas ou mais dimensão
- Criada com o comando: `array()` ou `matrix()`

`array(data = NA, dim = c(dim01,dim02,...), dimnames = NULL)`

`matrix(data = NA, nrow = 1, ncol = 1, byrow = FALSE, dimnames = NULL)`

- Acessar os elementos utilizando: `[<index01>,<index02>,... ]`
- Operadores *range* para cada dimensão individual
- Linhas e colunas podem ser nomeadas
- Números: **diferença significativa de performance para procedimentos numéricos**

# Conceitos Básicos – Estruturas de Dados

```
## Inicialização com um único valor
```

```
> m2d = array(1,c(2,3))
```

```
> m2d
```

```
      [,1] [,2] [,3]
[1,]     1     1     1
[2,]     1     1     1
```

```
## Inicialização com todos os valores
```

```
> m2d = array(1:6,c(2,3))
```

```
> m2d
```

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

```
## Inicialização com rotação
```

```
> m2d = array(1:2,c(2,3))
```

```
> m2d
```

```
      [,1] [,2] [,3]
[1,]     1     1     1
[2,]     2     2     2
```

```
## Inicialização com 'NA'
```

```
> m2d = array(NA,c(2,3))
```

```
> m2d
```

```
      [,1] [,2] [,3]
[1,]    NA    NA    NA
[2,]    NA    NA    NA
```

# Conceitos Básicos – Estruturas de Dados

## • Arrays - Indexação

```
> m2d = array(1:12,c(4,3))
> m2d
```

```
      [,1] [,2] [,3]
[1,]     1     5     9
[2,]     2     6    10
[3,]     3     7    11
[4,]     4     8    12
```

```
## Apenas as linhas 1 e 2
```

```
> m2d[c(1,2),]
```

```
      [,1] [,2] [,3]
[1,]     1     5     9
[2,]     2     6    10
```

```
## linhas 2 e 3, colunas 2 e 3
```

```
> m2d[c(2,3),c(2,3)]
```

```
      [,1] [,2]
[1,]     6    10
[2,]     7    11
```

```
## Apenas a linha 2
```

```
> m2d[2,]
```

```
[1]  2  6 10
```

```
## Apenas a coluna 2
```

```
> m2d[,2]
```

```
[1] 5 6 7 8
```

```
## Linhas com elementos da coluna 2 maiores que 6
```

```
> m2d[m2d[,2]>6,]
```

```
      [,1] [,2] [,3]
[1,]     3     7    11
[2,]     4     8    12
```

# Conceitos Básicos – Estruturas de Dados

## • Arrays – Indexação e Dimensões

```
> m2d = array(1:12,c(4,3))
```

```
> m2d
```

```
      [,1] [,2] [,3]
[1,]     1     5     9
[2,]     2     6    10
[3,]     3     7    11
[4,]     4     8    12
```

```
## Apenas as linhas 1 e 2
```

```
> m2d[c(1,2),]
```

```
      [,1] [,2] [,3]
[1,]     1     5     9
[2,]     2     6    10
```

```
## linhas 2 e 3, colunas 2 e 3
```

```
> m2d[c(2,3),c(2,3)]
```

```
      [,1] [,2]
[1,]     6    10
[2,]     7    11
```

```
## Apenas a linha 2
```

```
> m2d[2,]
```

```
[1]  2  6 10
```

```
## Apenas a coluna 2
```

```
> m2d[,2]
```

```
[1]  5  6  7  8
```

```
## Linhas com elementos da coluna 2 maiores que 6
```

```
> m2d[m2d[,2]>6,]
```

```
      [,1] [,2] [,3]
[1,]     3     7    11
[2,]     4     8    12
```

```
## Dimensões do Array
```

```
> dim(m2d)
```

```
[1]  4  3
```



# Conceitos Básicos – Estruturas de Dados

- Arrays – Nome das Dimensões

```
> x = array(1:6,c(2,3))
> rownames(x) = c("Linha01","Linha02");
> colnames(x) = c("Col01","Col02","Col03");
```

```
> x
```

	Col01	Col02	Col03
Linha01	1	3	5
Linha02	2	4	6

```
> x["Linha01",c("Col02","Col03")]
```

Col02	Col03
3	5

```
> x["Linha02",]
```

Col01	Col02	Col03
2	4	6

# Conceitos Básicos – Operações com Vetores

- Operações aritméticas com vetores e matrizes

```
# Potência, soma e produto de vetores
```

```
> x = c(1:5)
```

```
> x^2
```

```
[1] 1 4 9 16 25
```

```
> y = c(11:15)
```

```
> x+y
```

```
[1] 12 14 16 18 20
```

```
> x^2+y
```

```
[1] 12 16 22 30 40
```

```
> x * y
```

```
[1] 11 24 39 56 75
```

```
> x + 5
```

```
[1] 6 7 8 9 10
```

```
## Produto matricial. Operador %*%
```

```
> m = array(1:10,c(2,5))
```

```
> m
```

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

```
> m %*% x
```

```
      [,1]
[1,]    95
[2,]   110
```

# Conceitos Básicos – Exercício

Utilizar a base de dados:

BBG-BMFBOVESPA-Equities-PX\_LAST.xlsx

e o script:

VALE5-vol.EXERCICIO.r

Para calcular a volatilidade de VALE5

# Conceitos Básicos – Exercício

Utilizar a base de dados:

BBG-BMFBOVESPA-Equities-PX\_LAST.xlsx

e o script:

Equities-covariancia.EXERCICIO.r

Para calcular a matriz de covariância das ações:

CSNA3, ELET6, PETR3, PETR4, VALE5, IBOV

# Conceitos Básicos – Estrutura de Dados

- Data Frames
- Estrutura de dados que armazena dados de tipo diferentes mas com a **mesma dimensões**

```
NomeDaDataFrame = data.frame(Nome=Valor,...)
```

- Acesso aos dados:

```
NomeDaDataFrame$NomeColuna
```

```
NomeDaDataFrame[[ 'NomeColuna' ]]
```

```
NomeDaDataFrame[Linha, 'Nome' /NúmeroColuna]
```

etc...

- Indexação e operações lógicas análogas a vetores e matrizes

# Conceitos Básicos – Estrutura de Dados

- Data Frames - Exemplo

```
> df = data.frame(numeros=1:5,
                  textos=c("um", "dois", "três", "quatro", "cinco"),
                  datas=as.Date('2010-01-01')+1:5)
```

```
> df
  numeros textos      datas
1       1     um 2010-01-02
2       2    dois 2010-01-03
3       3   três 2010-01-04
4       4 quatro 2010-01-05
5       5   cinco 2010-01-06
```

```
> df[[1]]
```

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

```
> df[['textos']]
```

```
[1] um     dois   três   quatro cinco
```

```
Levels: cinco dois quatro três um
```

```
> df[c(2,5), 'textos']
```

```
[1] dois   cinco
```

```
Levels: cinco dois quatro três um
```

```
> df[2:5, 'textos']
```

```
[1] dois   três   quatro cinco
```

```
Levels: cinco dois quatro três um
```

```
> df[df[['datas']]>as.Date('2010-01-04'),]
```

```
  numeros textos      datas
4       4 quatro 2010-01-05
5       5   cinco 2010-01-06
```

# Conceitos Básicos – Exercício

Utilizar a base de dados:

BBG-BMFBOVESPA-Equities-PX\_LAST.xlsx

e o script:

Equities-covariância-data.frame.EXERCICIO.r

Para calcular a matriz de covariância das ações:

CSNA3, ELET6, PETR3, PETR4, VALE5, IBOV

# Conceitos Básicos – Estrutura de Dados

- List

- Estrutura de dados que armazena dados de tipo e dimensões diferentes

```
NomeDaLista = list(Nome=Valor,...)
```

- Acesso aos dados:

```
NomeDaLista$Nome
```

```
NomeDaLista[[ 'Nome' ]]
```

```
> list01 = list(a=1:10, d=Sys.Date(), m=array(1:6,c(2,3)))
```

```
> list01
```

```
$a
```

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

```
$d
```

```
[1] "2015-01-19"
```

```
$m
```

```
      [,1] [,2] [,3]
```

```
[1,]     1     3     5
```

```
[2,]     2     4     6
```

```
> list01$m[2,c(2,3)]
```

```
[1] 4 6
```

```
> list01[['m']][2,c(2,3)]
```

```
[1] 4 6
```



# Conceitos Básicos – Estrutura de Dados

- List: Exemplo de Aplicação
  - Armazenar dados de configurações:
    - Parâmetros de Modelos
    - Nomes de arquivos e colunas de dados
    - Endereço/Servidor e autenticação em bases de dados

## Exemplo:

- Configuração de acesso a um banco de dados

```
database = list(url='sqlserver01',
               schemna='MyDatabase',
               username="ReadOnly",
               passwd='ReadOnly');
```

- Configuração para leitura de um arquivo de dados

```
datafile01 = list(path='c:/tmp/file01.dat',
                  type='csv',
                  dec='.',
                  colnames = c('DATE', 'VALE5', 'PETR4', , 'GGBR4'));
```

# Conceitos Básicos – Funções

- Chamada de funções:

```
NomeDaFuncao (Arg01, Arg02, ...);
```

ou

```
NomeDaFuncao (Nome01 = Arg01, ..., NomeX=ArgX, ...);
```

- Declaração de função:

```
NomeDaFuncao = function(<Lista de Argumentos>) {
    <Comandos>
    return (<ListaDeValorDeRetorno>);
}
```

# Conceitos Básicos – Funções

Exemplo da chamada da função *log*:

```
log(x, base = exp(1));
```

Onde: 2 argumentos:

**x** obrigatório

**base** opcional, se o valor da não for indicado a base default é  $e=2,7182\dots$

```
## Exemplo: log(x, base = exp(1))
## Chama a função log passando um
x = log(100);
print(x); # [1] 4.60517
```

```
## Não nomeia os argumentos
x = log(100,10);
print(x); # [1] 2
```

```
## Nomeia os argumentos
x = log(100,base=10);
print(x); # [1] 2
```

# Conceitos Básicos – Funções

Exercício:

Construir a função `raiz(.)`:

```
raiz(x, radix = 2);
```

Onde: 2 argumentos:

`x` obrigatório

`radix` opcional, se o valor da não for indicado a base default é 2

# Conceitos Básicos – Funções

**Solução:** Função `raiz(x, radix = 2)`;

Onde: 2 argumentos:

`x` obrigatório

`radix` opcional, se o valor da não for indicado a base default é 2

```
## Definição de função
```

```
raiz = function(x, radix=2) {
```

```
  y = x^(1/radix);
```

```
  return(y)
```

```
}
```

```
## Exemplo execução
```

```
x = raiz(64);
```

```
print(x); # [1] 8
```

```
x = raiz(64,radix=6);
```

```
print(x); # [1] 2
```

# Conceitos Básicos – Funções

## Exercício 01:

Utilizar a base de dados:

BBG-BMFBOVESPA-Equities-PX\_LAST.xlsx

Construir a função `CalculaCovariancia(...)`:

```
CalculaCovariancia(pathWorkbook, codes);
```

Argumentos:

`pathWorkbook`: path da planilha de dados

`codes` : lista de códigos das ações

Retorno: Matriz de covariância para as ações selecionadas

# Conceitos Básicos – Funções

Problemas na modelagem da função:

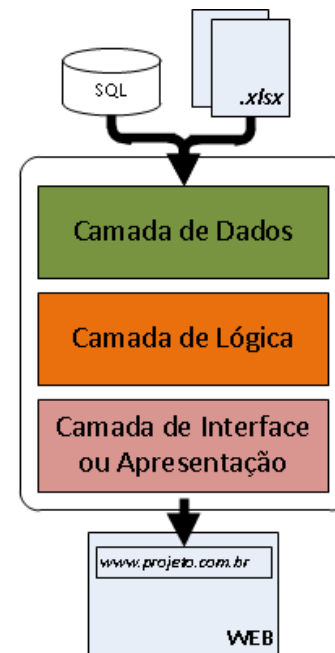
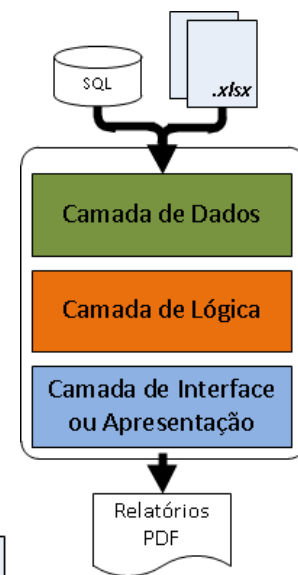
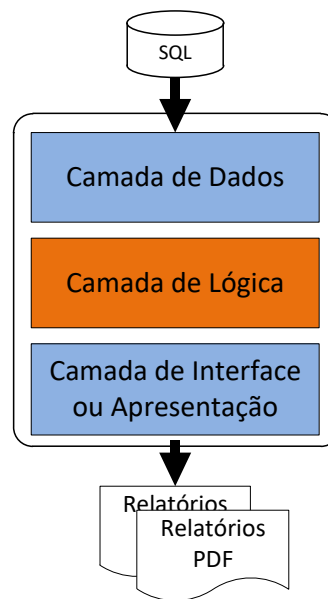
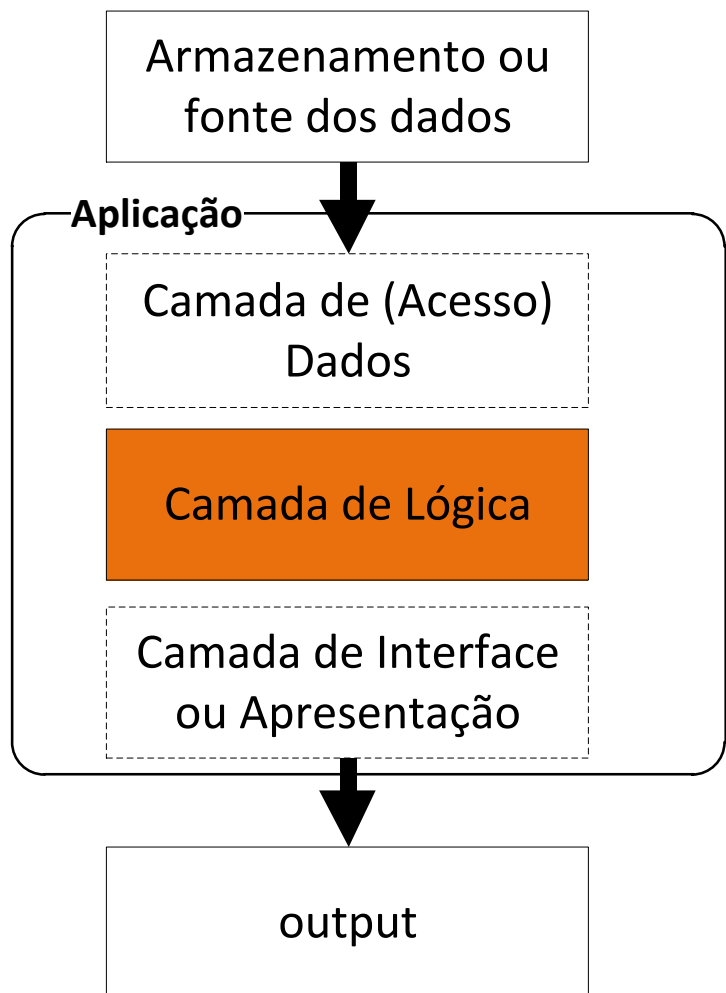
- Mudar o nome da planilha (ou a posição da planilha)
  - Alterar a posição dos dados na planilha
  - Alterar a forma de armazenamento dos dados
- ...entre outros.

Melhor solução:

- “Cada função deve ter uma função específica” e um programa (ou script chama as funções.

# Modelagem Quantitativa - Arquitetura

## Modelo de 3 Camadas





# Conceitos Básicos – Funções

**Exercício 02:** Construir as funções nos arquivo indicados

Utilizar a base de dados:

BBG-BMFBOVESPA-Equities-PX\_LAST.xlsx

Arquivo: lib/DataLoader.r

```
XLDataLoader (pathWorkbook)
```

Argumentos:

pathWorkbook: path da planilha de dados

Retorno: prices : Matriz de histórico de preços

Arquivo: lib/DataAnalysis.r

```
CalculaCovariancia (prices, codes);
```

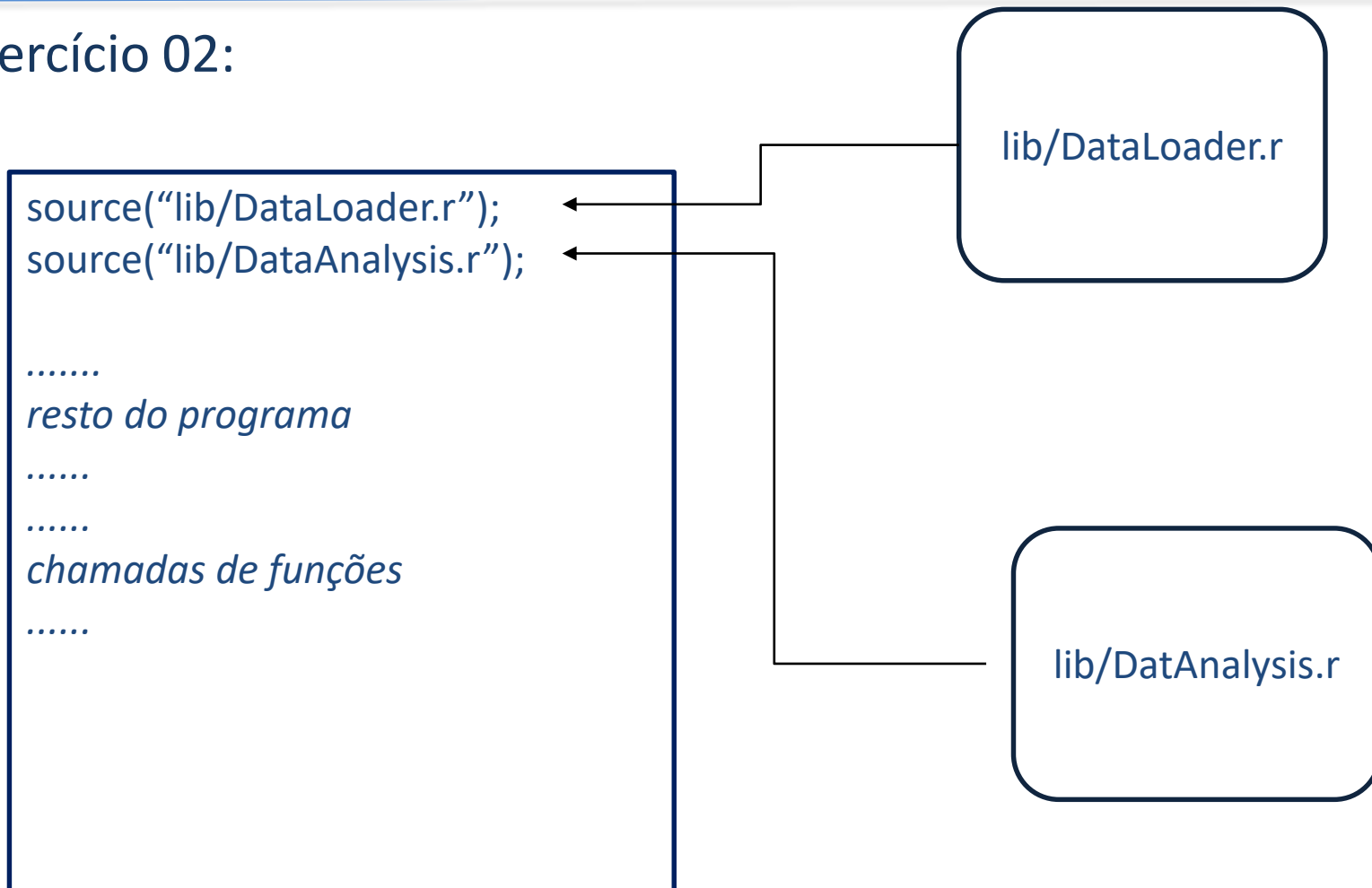
codes : lista de códigos das ações

prices : Matriz de histórico de preços

Retorno: Matriz de covariância

# Conceitos Básicos – Funções

## Exercício 02:



# Conceitos Básicos – Funções

**Exercício 03:** Construir as funções nos arquivo indicados

**Retorno:** `prices` : Matriz de histórico de preços

**Arquivo:** `lib/DataAnalysis.r`

```
CalculaCovariancia(prices, codes,
                    from=1, interval=NULL);
```

`codes` : lista de códigos das ações

`prices` : Matriz de histórico de preços

`from` : (opcional) Índice de início do intervalo de tempo

`interval` : (opcional) número de observações, se NULL utilizar o restante

**Retorno:** Matriz de covariância

# Conceitos Básicos – Estruturas de Dados

## Comandos para criar sequencias e vetores:

### - Repetição:

```
rep(x, ...)
```

```
rep.int(x, times)
```

```
> x = rep(c(1:3),10)
```

```
> x
[1] 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3
```

```
> x = rep(c(1:3),5)
```

```
> x
[1] 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3
```

```
> d = rep(c("um","dois"),3)
```

```
> d
[1] "um" "dois" "um" "dois" "um" "dois"
```

# Conceitos Básicos – Estruturas de Dados

## Comandos para criar sequencias e vetores:

### - Sequências:

```
seq(...)
```

```
seq(from = ..., to = ..., by =...)
```

```
> x = seq(1,10,by=2)
```

```
> x
```

```
[1] 1 3 5 7 9
```

```
> x = seq(from=as.Date('2010-01-05'),to=as.Date('2010-08-05'), by = "2 month")
```

```
> x
```

```
[1] "2010-01-05" "2010-03-05" "2010-05-05" "2010-07-05"
```

# Exercícios

1. Escrever uma função SomaUm que recebe um número  $x$  e retorna a soma com 1
2. Escrever uma função SomaN que recebe um número  $x$  e um número  $n$  e retorna a soma com  $n$  por default  $n=1$
3. Escrever uma função SomaProdutoN que recebe um número  $n$  e retorna a soma e o produto com  $n$  por default  $n=1$
4. Escrever uma função ValorPresenteJurosSimples.

## Inputs:

valor futuro (VF), taxa juros simples ( $r$ ), data inicial (dataInicial) opcional com *default* a data corrente e uma data final (dataFinal)

## Valor de retorno:

valor presente descontado pela taxa de juros. Considerar taxa de juros simples e dias corridos

# Exercícios

5. Escrever uma função ValorPresenteFluxo:

## Inputs:

Vetor de Fluxo de Caixa (CF: *Cash Flow*), vetor taxa juros exponencial 252 ( $r$ ), Vetor de dias úteis (DU)

## Valor de retorno:

Valor presente do fluxo de caixa descontado pela taxa de juros considerando capitalização composta por dias úteis.

Considere que se a taxa de juros for um único valor, replicar o mesmo valor para todas os fluxos