

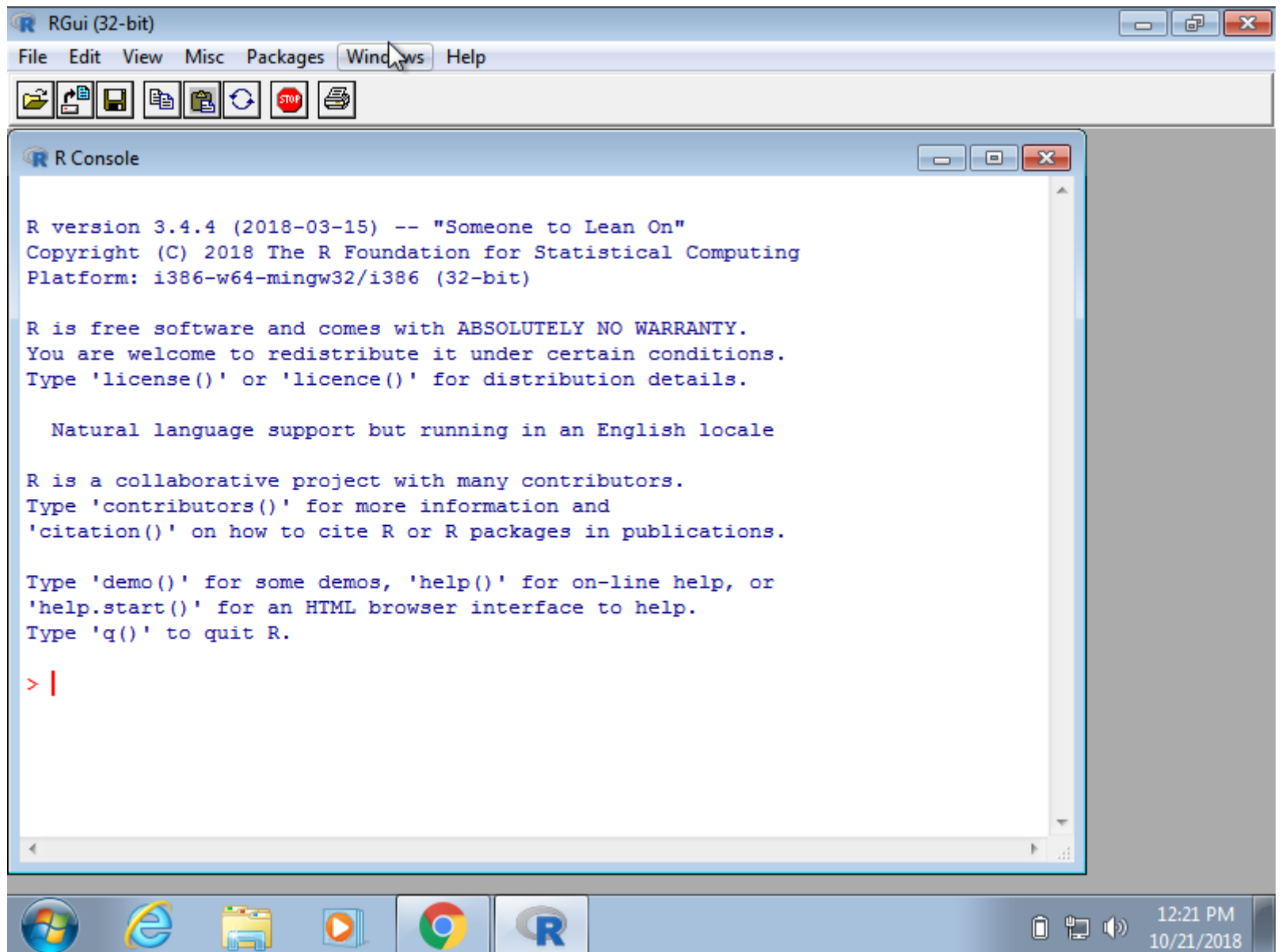


## R como calculadora (Revisão)

A linguagem do R é um tanto quanto intuitiva, muita coisa sai do jeito certo no chute! Para ver um exemplo disso, é interessante começar fazendo do R uma grande calculadora. Tente jogar no console  $2*2 - (4 + 4)/2$ . Pronto. Com essa simples expressão você já é capaz de imaginar (e certamente) como pedir ao R para fazer qualquer tipo de operação aritmética. Lição aprendida!

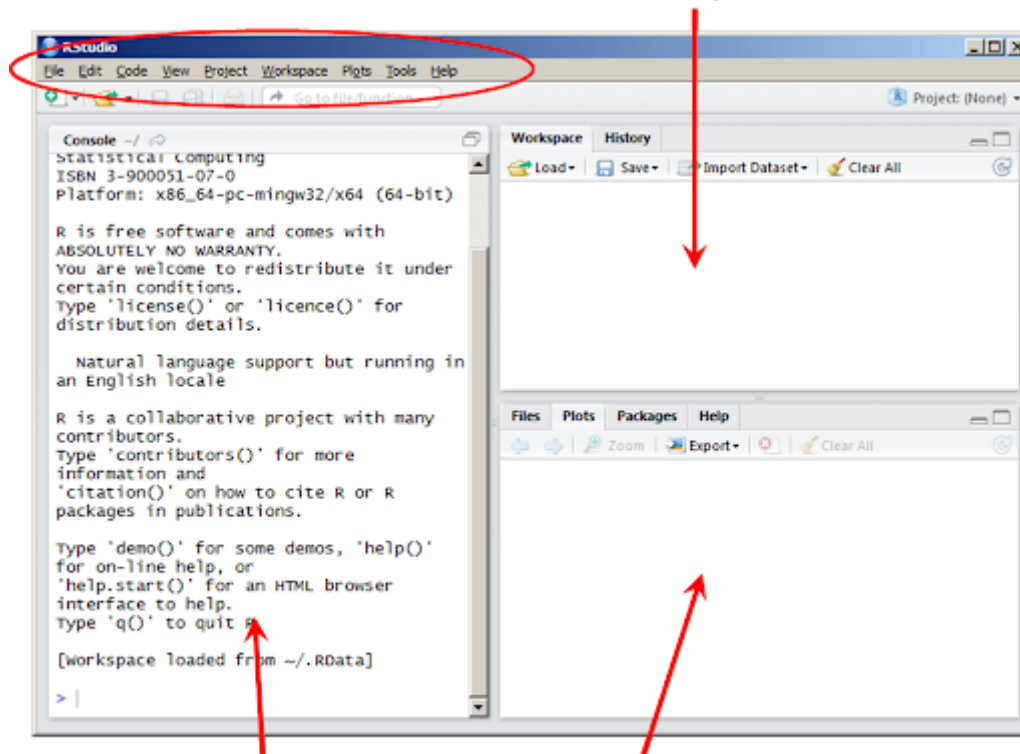
Além do mais, as operações e suas precedências são mantidas como na matemática, ou seja, divisão e multiplicação são calculadas antes da adição e subtração. E os parênteses nunca são demais!

Falando em matemática, o  $\pi$  já vem definido!



Menu Items

Workspace



R Console

Plots

<https://rdr.io/snippets/> (<https://rdr.io/snippets/>).

In [117]:

```
pi
```

3.14159265358979

In [118]:


```
sin(pi/2) + 2*3
```

7

In [119]:

```
(sin(pi/2) + 2)*3
```

9

Veja que apareceu a função `sin()` , o famoso seno. Tente outras funções trigonométricas para ver se acerta de prima! 

Mas a ideia é essa. Ser intuitivo. A dica aqui é tentar! No R, um erro não é nenhuma dor de cabeça, é rápido consertar e testar (debugar, para os mais íntimos).

## Objetos atômicos

Existem cinco classes básicas ou “atômicas” no R:

- character
- numeric
- integer
- logical



## characters ☐

In [37]:

```
"a"
```

```
'a'
```

In [38]:

```
"1"
```

```
'1'
```

In [39]:

```
"positivo"
```

```
'positivo'
```

In [40]:

```
"Error: objeto x não encontrado"
```

```
'Error: objeto x não encontrado'
```

numeric 

In [41]:

1

1

In [42]:

0.10

0.1

In [43]:

0.95

0.95

In [44]:

pi

3.14159265358979

# integer

In [45]:

```
1L
```

1

In [46]:

```
5L
```

5

In [47]:

```
10L
```

10

# logical

In [48]: **TRUE**

TRUE

In [50]: **FALSE**

FALSE

Para saber a classe de um objetivo, você pode usar a função `class()` .

```
In [51]: x <- 1  
         class(x)
```

'numeric'

```
In [52]: y <- "a"  
         class(y)
```

'character'

```
In [53]: z <- TRUE  
         class(z)
```

'logical'

## Operadores aritméticos $+$ $\div$ $-$ $\times$

Os bons e conhecidos operadores aritméticos. Com números reais eles funcionam como na matemática, mas ao saírmos da reta, eles podem fazer muito mais!

Operador	Descrição
$x + y$	Adição de x com y
$x - y$	Subtração de y em x
$x * y$	Multiplicação de x e y
$x / y$	Divisão de x por y
$x^y$ ou $x**y$	x elevado a y-ésima potência
$x\%y$	Resto da divisão de x por y (módulo)
$x\%/\%y$	Parte inteira da divisão de x por y

Exemplos:

In [120]:

```
1 + 1
```

2

In [121]:

```
10-8
```

2

In [122]:

```
2*10
```

20



In [123]:

```
18/3
```

6

In [124]:

```
2^4
```

16

In [125]:

```
2**4
```

16

In [126]:

```
9%%2
```

1

In [127]:

```
9/%2
```

4

## Operadores lógicos

Operadores lógicos retornarão sempre ou TRUE ou FALSE. Eles definem perguntas que aceitam apenas verdadeiro e falso como resposta, como sugere o quadro abaixo.

operador	descricao
$x < y$	x menor que y?
$x \leq y$	x menor ou igual a y?
$x > y$	x maior que y?
$x \geq y$	x maior ou igual a y?
$x == y$	x igual a y?
$x != y$	x diferente de y?
$!x$	Negativa de x
$x$	y
$x \& y$	x e y são verdadeiros?
$xor(x, y)$	x ou y são verdadeiros (apenas um deles)?

In [128]:

```
1 < 1
```

FALSE

In [129]:

```
1 <= 1
```

TRUE

In [130]:

```
1 == 0.999
```

FALSE

In [131]:

```
1 == 0.9999999999999999
```

TRUE

In [132]: 13.5 != 13.5

FALSE

In [133]: !TRUE

FALSE

In [134]: TRUE & FALSE

FALSE

In [135]: TRUE & TRUE

TRUE

In [136]: `xor(TRUE, TRUE)`

FALSE

In [137]: `xor(TRUE, FALSE)`

TRUE

Não se preocupe se você não entendeu muito bem o operador *xor* ele não será empregado neste curso, mas é importante em algumas áreas como Lógica e Inteligência artificial. 😊

💡 Agora um conceito importante da equivalência numérica de objetos lógicos em lógica e em computação: `TRUE` e `1` são equivalentes assim como `FALSE` e `0`.

```
In [138]: TRUE == 1
```

```
TRUE
```

```
In [139]: TRUE == 2
```

```
FALSE
```

```
In [140]: FALSE == 0
```

```
TRUE
```



## VETORES

Vetores no R são os objetos mais simples que podem guardar objetos atômicos.

```
In [141]: vetor1 <- c(1, 2, 3, 4)
          vetor2 <- c("a", "b", "c")
```

```
In [142]: vetor1
```

```
1 2 3 4
```

```
In [143]: vetor2
```

```
"a" "b" "c"
```

Um vetor tem sempre a mesma classe dos objetos que guarda.

```
In [144]: class(vetor1)
```

```
'numeric'
```

```
In [145]: class(vetor2)
```

```
'character'
```

De forma bastante intuitiva, você pode fazer operações com vetores.

In [146]:

```
vetor1-1
```

0 · 1 · 2 · 3

Lembrando do vetor1

In [147]:

```
vetor1
```

1 · 2 · 3 · 4

Quando você faz `vetor1 - 1`, o R subtrai 1 de cada um dos elementos do vetor. O mesmo acontece quando você faz qualquer operação aritmética com vetores no R. Vamos ver outros exemplos

```
In [148]: vetor1 / 2
```

0.5 · 1 · 1.5 · 2

```
In [149]: vetor1 * 10
```

10 · 20 · 30 · 40

Você também pode fazer operações que envolvem mais de um vetor:

```
In [150]: vetor1 * vetor1
```

1 · 4 · 9 · 16

## MISTURANDO OBJETOS

*Vetores são homogêneos. Os elementos de um vetor são sempre da mesma classe. Ou todos são numéricos, ou são todos character, ou todos são lógicos etc. Não dá para ter um número e um character no mesmo vetor, por exemplo.*

Se colocarmos duas ou mais classes diferentes dentro de um mesmo vetor, o R vai forçar que todos os elementos passem a pertencer à mesma classe. O número 1.7 viraria "1.7" se fosse colocado ao lado de um "a".

```
In [151]: y <- c(1.7, "a") # character  
class(y)
```

'character'

```
In [152]: y <- c(TRUE, 2) # numeric  
class(y)
```

'numeric'

```
In [153]: y <- c(TRUE, 'a') # numeric  
class(y)
```

'character'

A ordem de precedência é:

**DOMINANTE** — *character* > *complex* > *numeric* > *integer* > *logical* — **RECESSIVO**



## Matrizes

Matrizes são vetores com duas dimensões (e por isso só possuem elementos de uma mesma classe).

```
In [154]: m <- matrix(1:6, nrow = 2, ncol = 3)
```

```
In [155]: m
```

A matrix:

2 × 3 of

type int

1	3	5
2	4	6

Observe que uma matriz é uma tabela neste caso temos:

- nrow = número de linhas
- ncol = número de colunas

Ou seja nossa matriz (tabela) tem seis posições duas linhas e três colunas

1	2	3
4	4	6

```
In [156]: # Observe o que o 1:6 faz  
1:6
```

1 · 2 · 3 · 4 · 5 · 6

E como seria o código se quiséssemos que a matriz fosse assim

1	2
3	4
5	6

Nesse caso quantas colunas tem a matriz

`ncols = 2`

E quantas linhas

`nrow=3`

```
In [157]: m = matrix(1:6, nrow = 3, ncol = 2)
m
```

A  
matrix:  
 $3 \times 2$   
of type  
int

1	4
2	5
3	6

## Listas

Listas são um tipo especial de vetor que aceita elementos de classes diferentes.

```
In [158]: x <- list(1:5, "Z", TRUE, c("a", "b"))  
x
```

```
1.      1 · 2 · 3 · 4 · 5  
2. 'Z'  
3. TRUE  
4.      'a' · 'b'
```

É um dos objetos mais importantes para armazenar dados e vale a pena saber manuseá-los bem. Existem muitas funções que fazem das listas objetos incrivelmente úteis.

# Criando uma Lista

Criamos uma lista com a função `list()`, que aceita um número arbitrário de elementos. Listas aceitam QUALQUER tipo de objeto. Podemos ter listas dentro de listas, por exemplo. Como para quase todas as classes de objetos no R, as funções `is.list()` e `as.list()` também existem.

Na lista pedido abaixo, temos `numeric`, `Date`, `character`, `vetor de character` e `list` contida em uma lista:

```
In [159]: pedido <- list(pedido_id = 8001406,
                        pedido_registro = as.Date("2017-05-25"),
                        nome = "Athos",
                        sobrenome = "Petri Damiani",
                        cpf = "12345678900",
                        email = "athos.damiani@gmail.com",
                        qualidades = c("incrível", "impressionante"),
                        itens = list(
                            list(descricao = "Ferrari",
                                frete = 0,
                                valor = 500000),
                            list(descricao = "Dolly",
                                frete = 1.5,
                                valor = 3.90)
                        ),
                        endereco = list(entrega = list(logradouro = "Rua da Glória",
                                                        numero = "123",
                                                        complemento = "apto 71"),
                                        cobranca = list(logradouro = "Rua Jose de Oliveira Coutin
ho",
                                                        numero = "151",
                                                        complemento = "5o andar")
                        )
                    )
```



Calma vamos estudar isso com bastante calma mais a frente agora a idéia é que você conheça todos os tipos de coleções de dados como vetores, matrizes e listas 😊

## Operações úteis

```
In [160]: pedido$cpf      # elemento chamado 'cpf'
```

```
'12345678900'
```

```
In [161]: pedido[1]      # nova lista com apenas o primeiro elemento
```

```
$pedido_id = 8001406
```

```
In [162]: pedido[[2]]    # segundo elemento
```

```
2017-05-25
```

```
In [163]: pedido["nome"] # nova lista com apenas o elemento chamado 'nome'
```

```
$nome = 'Athos'
```

Certamente você se deparará com listas quando for fazer análise de dados com o R. Nos tópicos mais aplicados, iremos aprofundar sobre o tema. O pacote `purrr` contribui com funcionalidades incríveis para listas.

## data.frame

Um data.frame é o mesmo que uma tabela do SQL ou um spreadsheet do Excel, por isso são objetos muito importantes.

Usualmente, seus dados serão importados para um objeto data.frame. Em grande parte do curso, eles serão o principal objeto de estudo.

Os `data.frame`'s são listas especiais em que todos os elementos possuem o mesmo comprimento. Cada elemento dessa lista pode ser pensado como uma coluna da tabela. Seu comprimento representa o número de linhas.

Já que são listas, essas colunas podem ser de classes diferentes. Essa é a grande diferença entre `data.frame`'s e matrizes. Algumas funções úteis:

`head()` - Mostra as primeiras 6 linhas.

`tail()` - Mostra as últimas 6 linhas.


`dim()` - Número de linhas e de colunas.

`names()` - Os nomes das colunas (variáveis).

`str()` - Estrutura do `data.frame`. Mostra, entre outras coisas, as classes de cada coluna.

`cbind()` - Acopla duas tabelas lado a lado.

`rbind()` - Empilha duas tabelas.

O exemplo abaixo mostra que uma lista pode virar data.frame se todos os elementos tiverem o mesmo comprimento. 

```
In [167]: minha_lista <- list(x = c(1, 2, 3), y = c("a", "b"))
as.data.frame(minha_lista)
```

Error in (function (..., row.names = NULL, check.rows = FALSE, check.names = TRUE, :  
arguments imply differing number of rows: 3, 2

Traceback:

```
1. as.data.frame(minha_lista)
2. as.data.frame.list(minha_lista)
3. do.call(data.frame, c(x, alis))
4. (function (..., row.names = NULL, check.rows = FALSE, check.names = TRUE,
.   fix.empty.names = TRUE, stringsAsFactors = default.stringsAsFactors())
. {
.   data.row.names <- if (check.rows && is.null(row.names))
.     function(current, new, i) {
.       if (is.character(current))
.         new <- as.character(new)
.       if (is.character(new))
.         current <- as.character(current)
.       if (anyDuplicated(new))
.         return(current)
.       if (is.null(current))
.         return(new)
.       if (all(current == new) || all(current == ""))
.         return(new)
.       stop(gettextf("mismatch of row names in arguments of 'data.frame', ite
m %d",
.         i), domain = NA)
.     }
.   else function(current, new, i) {
.     if (is.null(current)) {
.       if (anyDuplicated(new)) {
.         warning(gettextf("some row.names duplicated: %s --> row.names NOT
used",
.           paste(which(duplicated(new)), collapse = ",")),
.           domain = NA)
.         current
```

```

.           }
.           else new
.       }
.       else current
.   }
.   object <- as.list(substitute(list(...)))[-1L]
.   mirn <- missing(row.names)
.   mrn <- is.null(row.names)
.   x <- list(...)
.   n <- length(x)
.   if (n < 1L) {
.       if (!mrn) {
.           if (is.object(row.names) || !is.integer(row.names))
.               row.names <- as.character(row.names)
.           if (anyNA(row.names))
.               stop("row names contain missing values")
.           if (anyDuplicated(row.names))
.               stop(gettextf("duplicate row.names: %s", paste(unique(row.names[du
plicated(row.names)]),
.                   collapse = ", ")), domain = NA)
.       }
.       else row.names <- integer()
.       return(structure(list(), names = character(), row.names = row.names,
.           class = "data.frame"))
.   }
.   vnames <- names(x)
.   if (length(vnames) != n)
.       vnames <- character(n)
.   no.vn <- !nzchar(vnames)
.   vlist <- vnames <- as.list(vnames)
.   nrow <- ncol <- integer(n)
.   for (i in seq_len(n)) {
.       xi <- if (is.character(x[[i]]) || is.list(x[[i]]))
.           as.data.frame(x[[i]], optional = TRUE, stringsAsFactors = stringsAsFac
tors)
.       else as.data.frame(x[[i]], optional = TRUE)
.       nrow[i] <- .row_names_info(xi)
.       ncol[i] <- length(xi)

```



```

.       namesi <- names(xi)
.       if (ncols[i] > 1L) {
.           if (length(namesi) == 0L)
.               namesi <- seq_len(ncols[i])
.           vnames[[i]] <- if (no.vn[i])
.               namesi
.           else paste(vnames[[i]], namesi, sep = ".")
.       }
.       else if (length(namesi)) {
.           vnames[[i]] <- namesi
.       }
.       else if (fix.empty.names && no.vn[[i]]) {
.           tmpname <- deparse(object[[i]], nlines = 1L)[1L]
.           if (startsWith(tmpname, "I(") && endsWith(tmpname,
.               ")")) {
.               ntmpn <- nchar(tmpname, "c")
.               tmpname <- substr(tmpname, 3L, ntmpn - 1L)
.           }
.           vnames[[i]] <- tmpname
.       }
.       if (mirn && nrows[i] > 0L) {
.           rowsi <- attr(xi, "row.names")
.           if (any(nzchar(rowsi)))
.               row.names <- data.row.names(row.names, rowsi,
.                   i)
.       }
.       nrows[i] <- abs(nrows[i])
.       vlist[[i]] <- xi
.   }
.   nr <- max(nrows)
.   for (i in seq_len(n)[nrows < nr]) {
.       xi <- vlist[[i]]
.       if (nrows[i] > 0L && (nr%%nrows[i] == 0L)) {
.           xi <- unclass(xi)
.           fixed <- TRUE
.           for (j in seq_along(xi)) {
.               xi1 <- xi[[j]]
.               if (is.vector(xi1) || is.factor(xi1))

```

```

.         xi[[j]] <- rep(xi1, length.out = nr)
.     else if (is.character(xi1) && inherits(xi1, "AsIs"))
.         xi[[j]] <- structure(rep(xi1, length.out = nr),
.             class = class(xi1))
.     else if (inherits(xi1, "Date") || inherits(xi1,
.         "POSIXct"))
.         xi[[j]] <- rep(xi1, length.out = nr)
.     else {
.         fixed <- FALSE
.         break
.     }
. }
. if (fixed) {
.     vlist[[i]] <- xi
.     next
. }
. }
. stop(gettextf("arguments imply differing number of rows: %s",
.     paste(unique(nrows), collapse = ", ")), domain = NA)
. }
. value <- unlist(vlist, recursive = FALSE, use.names = FALSE)
. vnames <- as.character(unlist(vnames[ncols > 0L]))
. if (fix.empty.names && any(noname <- !nzchar(vnames)))
.     vnames[noname] <- paste0("Var.", seq_along(vnames))[noname]
. if (check.names) {
.     if (fix.empty.names)
.         vnames <- make.names(vnames, unique = TRUE)
.     else {
.         nz <- nzchar(vnames)
.         vnames[nz] <- make.names(vnames[nz], unique = TRUE)
.     }
. }
. }
. names(value) <- vnames
. if (!mrn) {
.     if (length(row.names) == 1L && nr != 1L) {
.         if (is.character(row.names))
.             row.names <- match(row.names, vnames, 0L)
.         if (length(row.names) != 1L || row.names < 1L ||

```

```

.         row.names > length(vnames))
.         stop("'row.names' should specify one of the variables")
.         i <- row.names
.         row.names <- value[[i]]
.         value <- value[-i]
.     }
.     else if (!is.null(row.names) && length(row.names) !=
.         nr)
.         stop("row names supplied are of the wrong length")
. }
. else if (!is.null(row.names) && length(row.names) != nr) {
.     warning("row names were found from a short variable and have been discarde
d")
.     row.names <- NULL
. }
. class(value) <- "data.frame"
. if (is.null(row.names))
.     attr(value, "row.names") <- .set_row_names(nr)
. else {
.     if (is.object(row.names) || !is.integer(row.names))
.         row.names <- as.character(row.names)
.     if (anyNA(row.names))
.         stop("row names contain missing values")
.     if (anyDuplicated(row.names))
.         stop(gettextf("duplicate row.names: %s", paste(unique(row.names[duplic
ated(row.names)])),
.             collapse = ", ")), domain = NA)
.     row.names(value) <- row.names
. }
. value
. }(x = c(1, 2, 3), y = c("a", "b"), check.names = TRUE, fix.empty.names = TRUE,
.     stringsAsFactors = FALSE)
5. stop(gettextf("arguments imply differing number of rows: %s",
.     paste(unique(nrows), collapse = ", ")), domain = NA)

```

# Fazendo do Jeito Certo

```
In [168]: minha_lista <- list(x = c(1, 2, 3), y = c("a", "b", 'c'))  
as.data.frame(minha_lista)
```

A data.frame: 3  
× 2

x	y
<dbl>	<chr>
1	a
2	b
3	c

In [ ]: