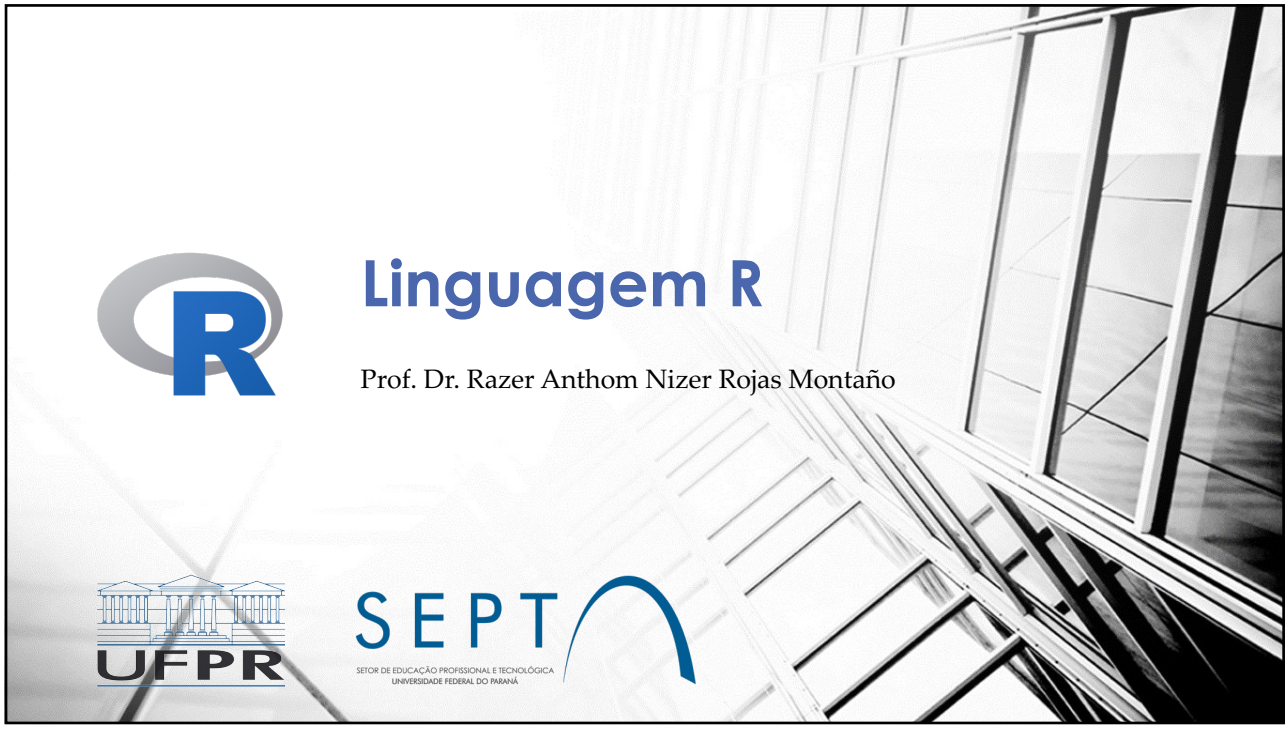
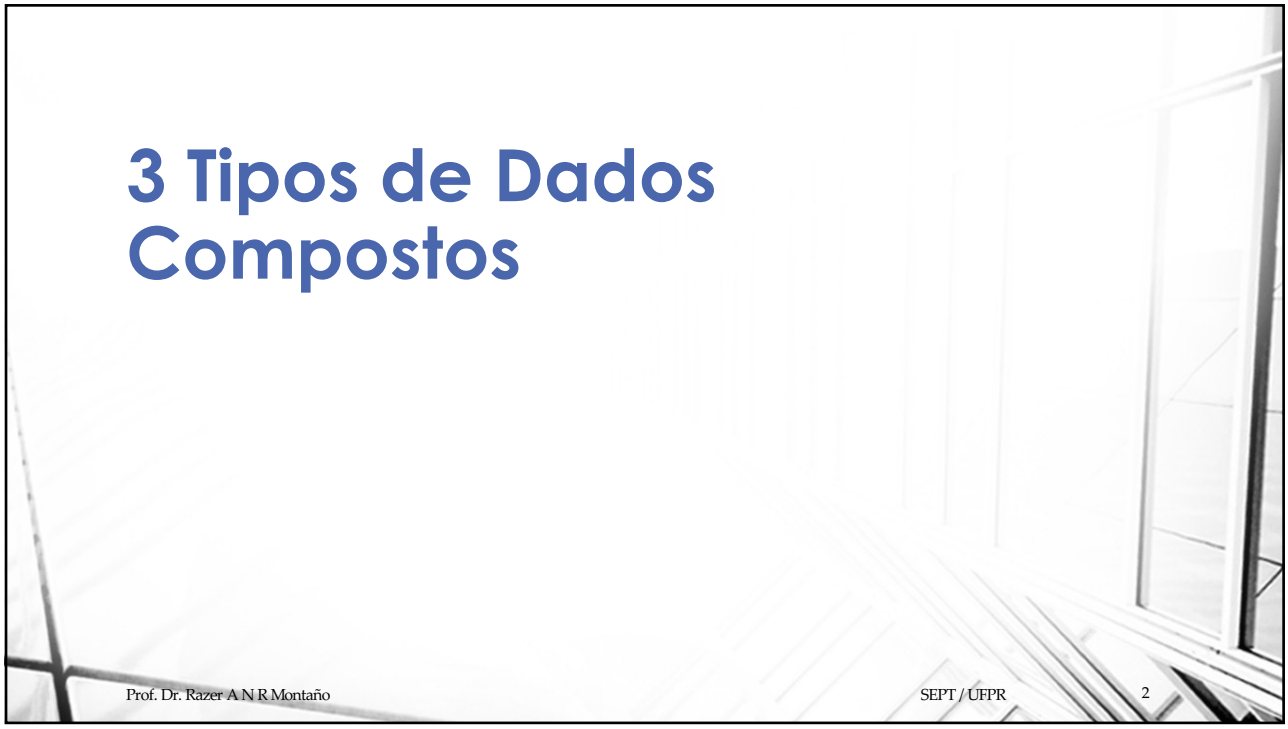

Linguagem R
Prof. Dr. Razer Anthom Nizer Rojas Montaño
UFPR
SEPT
SETOR DE EDUCAÇÃO PROFISSIONAL E TECNOLÓGICA
UNIVERSIDADE FEDERAL DO PARANÁ

1


3 Tipos de Dados Compostos
Prof. Dr. Razer A N R Montaño
SEPT / UFPR
2

2

Tipos de Dados Compostos.

- Tipos de dados compostos
  - **(3.1) Vetores** : coleção de elementos atômicos, do mesmo tipo, unidimensional
  - **(3.2) Matrizes** : Um vetor com várias dimensões
  - **(3.3) Fatores** : Usados para representar valores categóricos (categorias)
  - **(3.3) Fórmulas** : Representa fórmulas, dependência entre variáveis
  - **(3.4) Listas** : Coleção de elementos, atômicos ou não, pode misturar tipos de dados
  - **(3.5) Data Frames** : Tipo especial de lista, onde cada elemento possui o mesmo tamanho, sendo portanto retangular

3

3 Tipos de Dados Compostos

3.1 Vetores

4

Vetores: Criação

- Podem guardar objetos atômicos
  - `character, complex, logical, integer, numeric`
- Pode-se criar um vetor vazio com `vector()` (default `logical`)

```
> x <- vector()
> x
logical(0)
```

- Pode-se definir o tipo e o tamanho do vetor

```
> x <- vector("integer", length=5)
> x
[1] 0 0 0 0 0
```


5

Vetores: Criação

- Pode-se criar vetores com as funções
  - `character(), complex(), logical(), integer(), numeric()`

```
> x <- integer(5)
> x
[1] 0 0 0 0 0
```

- Outro exemplo

```
> x <- character(10)
> x
[1] "" "" "" "" "" "" "" "" "" ""
```


6

Vetores: Criação

- Pode-se criar um vetor indicando seu conteúdo usando a função
  - `c()`
- Exemplo

```
> vetor1 <- c(1, 2, 3, 4)
> vetor1
[1] 1 2 3 4
> class(vetor1)
[1] "numeric"
> typeof(vetor1)
[1] "double"
```


7

Vetores: Criação

- Se quiser forçar o uso de inteiros acrescente `L`

```
> vetor1 <- c(1L, 2L, 3L, 4L)
> vetor1
[1] 1 2 3 4
> class(vetor1)
[1] "integer"
> typeof(vetor1)
[1] "integer"
```


8

Vetores: Criação

- Outro exemplo de uso da função `c()`

```
> vetor2 <- c("a", "b", "c", "d")
> vetor2
[1] "a" "b" "c" "d"
```


9

Vetores: Criação

- Pode-se criar vetores a partir de uma sequência ou repetição de valores
  - Usando operador `:`
    - Sequência de valores de 1 em 1
  - Usando função `seq()`
    - Sequência qualquer de valores
  - Usando função `rep()`
    - Números repetidos

10

Vetores: Criação

- Usando operador :
  - Cria uma sequência de valores (de 1 em 1)
  - Crescente ou decrescente
  - Pode conter números negativos

```
> vet <- 1:10
> vet
 [1]  1  2  3  4  5  6  7  8  9 10
> vet <- 7:1
> vet
 [1] 7 6 5 4 3 2 1
> vet <- -3:1
> vet
 [1] -3 -2 -1  0  1
```


11

Vetores: Criação

- Usando a função `seq()`
  - Cria uma sequência de valores, com qualquer tipo de passo
  - Crescente ou decrescente
  - Pode usar outros vetores de "exemplo"
- Protótipo

```
seq(from = 1, to = 1, by = ((to - from)/(length.out - 1)),
length.out = NULL, along.with = NULL, ...)
```

- Onde:
  - `from`: início da sequência
  - `to`: fim da sequência
  - `by`: passo de incremento da sequência (default 1)
  - `length.out`: tamanho da sequência, quando não se sabe o passo a ser dado, mas se quer uma sequência de determinado tamanho
  - `along.with`: obtém o tamanho da sequência a partir do tamanho do elemento passado aqui

12

Vetores: Criação

- Gera uma sequência de 1 a 10

```
> vet <- seq(10)
> vet
 [1]  1  2  3  4  5  6  7  8  9 10
```

- Gera uma sequência de 1 a -5

```
> vet <- seq(-5)
> vet
 [1]  1  0 -1 -2 -3 -4 -5
```

- Gera uma sequência de 2 a 7

```
> vet <- seq(2, 7)
> vet
 [1] 2 3 4 5 6 7
```


13

Vetores: Criação

- Sequências geradas com valores decimais

```
> seq(1.5, 3.5, 0.5)
[1] 1.5 2.0 2.5 3.0 3.5
> seq(4.5, 2.0, -0.5)
[1] 4.5 4.0 3.5 3.0 2.5 2.0
```


14

Vetores: Criação

- Gera uma sequência de 1 a 10, de 2 em 2

```
> vet <- seq(1, 10, 2)
> vet
 [1] 1 3 5 7 9
```

- Gera uma sequência de 1 a 4, de 0.5 em 0.5

```
> vet <- seq(1, 4, 0.5)
> vet
 [1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0
```

- Gera uma sequência de 1 a 4, contendo 10 elementos

```
> vet <- seq(1, 4, length.out=10)
> vet
 [1] 1.000000 1.333333 1.666667 2.000000 2.333333 2.666667
3.000000 3.333333 3.666667 4.000000
```


15

Vetores: Criação

- Gera uma sequência de 1 a 10, sendo o tamanho do vetor resultante igual ao tamanho do vetor passado em `along.with`

```
> vetor <- c(4, 1, 19, 3, 15)
> vet <- seq(1, 10, along.with=vetor)
> vet
 [1]  1.00  3.25  5.50  7.75 10.00
```

- Uma função mais fácil é a `seq_along()`, que gera uma sequência começando em 1 até o número de elementos do parâmetro passado, de 1 em 1

```
> vetor <- c(4, 1, 19, 3, 15)
> seq_along(vetor)
 [1] 1 2 3 4 5
```


16

Vetores: Criação

- Usando a função `rep()`
  - Repete os valores ou vetores
- Protótipo

```
rep(x, times, length.out, each)
```

- Onde:
  - `x`: elemento a ser repetido, pode ser um vetor
  - `times`: quantas vezes será repetido (default 1), pode receber um vetor para indicar quantidades diferentes de repetições
  - `length.out`: tamanho da sequência, quando não se sabe quantas vezes repetir, mas se quer uma quantidade igual à do parâmetro passado
  - `each`: cada elemento de x é repetido `each` vezes

17

Vetores: Criação

- Repete 10, 3 vezes

```
> rep(10, 3)
 [1] 10 10 10
```

- Repete o vetor 4 vezes

```
> rep(5:8, 4)
 [1] 5 6 7 8 5 6 7 8 5 6 7 8 5 6 7 8
```

- Repete o vetor até que o tamanho do resultado seja 7

```
> rep(5:6, length.out=7)
 [1] 5 6 5 6 5 6 5
```


18

Vetores: Criação

- Pode-se repetir os elementos de um vetor de forma diferente

```
> rep(c(10, 20), times=c(2, 4))
 [1] 10 10 20 20 20 20
> rep(c(1,2), times=c(5,3))
[1] 1 1 1 1 1 2 2 2
```


19

Vetores: Criação.

- Criar vetor com elementos repetidos: `rep`

```
> rep(10, 5)
[1] 10 10 10 10 10
> rep(c(1,2), 3)
[1] 1 2 1 2 1 2
> rep(c(1,2), each=3)
[1] 1 1 1 2 2 2
```


20

Vetores: Coerção

- Se forem usados tipos diferentes na função `c()`, R faz a coerção dos tipos no seguinte sentido

- Exemplo

```
> vet <- c(TRUE, 1L, 10.2, "razer")
> vet
[1] "TRUE"  "1"     "10.2"  "razer"
```

- Na coerção de `logical` para um tipo numérico:
  - `TRUE` → `1`
  - `FALSE` → `0`

Vetores: Coerção.

- As funções `as.xxxx()` convertem todos os elementos do vetor
- Exemplo

```
> x <- 1:3
> class(x)
[1] "integer"
> as.character(x)
[1] "1" "2" "3"
```

Vetores: Manipulação

- Indexação: Acesso aos dados de um vetor
  - Inicia em 1
  - Operador `[]`
- Exemplo

```
> vetor3 <- 1:10
> vetor3
 [1]  1  2  3  4  5  6  7  8  9 10
> vetor3[5]
[1] 5
> vetor3[5] <- 777
> vetor3
 [1]   1   2   3   4 777   6   7   8   9  10
```


Vetores: Manipulação

- Pode-se retornar mais de um elemento
  - Usa-se `c()` dentro da indexação do vetor
- Exemplo

```
> vetor <- 15:24
> vetor
 [1] 15 16 17 18 19 20 21 22 23 24
> vetor[c(2, 5)]
[1] 16 19
```

Vetores: Manipulação

- Para adicionar um elemento ou vetor: `append`

```
> x <- 1:3
> y <- 4:6
> x
[1] 1 2 3
> y
[1] 4 5 6
> append(x, y)
[1] 1 2 3 4 5 6
```


25

Vetores: Manipulação

- Para remover um elemento do vetor, usa-se o **índice negativo**

```
> x <- 20:30
> x
 [1] 20 21 22 23 24 25 26 27 28 29 30
> x[3]
[1] 22
> x[-3]
 [1] 20 21 23 24 25 26 27 28 29 30
```


26

Vetores: Manipulação

- Para remover os elementos em uma **faixa de valores de índices**

```
> x <- 50:70

> x

 [1] 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66
67 68 69 70

> x[-(5:10)]

 [1] 50 51 52 53 60 61 62 63 64 65 66 67 68 69 70
```


27

Vetores: Manipulação

- Para remover um elemento pelo seu **valor**

```
> x <- 50:70

> x

 [1] 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66
67 68 69 70

> x[x!=51]

 [1] 50 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68
69 70
```


28

Vetores: Manipulação.

- Para remover elementos por uma **lista de valores**

```
> x <- 50:70
> x
 [1] 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66
67 68 69 70
> x[!x %in% c(53, 55, 66)]
 [1] 50 51 52 54 56 57 58 59 60 61 62 63 64 65 67 68 69 70
```


29

Vetores: Tamanho.

- Obter Tamanho do vetor : `length(vet)`
- Obter o último elemento : `vet[length(vet)]`

```
> x <- 20:30
> x
 [1] 20 21 22 23 24 25 26 27 28 29 30
> length(x)
[1] 11
> x[length(x)]
[1] 30
```


30

Vetores: Operações

- Consegue-se fazer operações com todos os elementos dos vetores de forma fácil
- Exemplo

```
> vetor1 <- c(1,2,3,4)
> vetor1 - 1
[1] 0 1 2 3
> x <- vetor1 * 10
> x
[1] 10 20 30 40
> vetor1 / 2
[1] 0.5 1.0 1.5 2.0
```


31

Vetores: Operações

- Também consegue-se fazer operações entre vetores
- Exemplo

```
> vetor1 * vetor1
[1]  1  4  9 16
> vetor2 <- c(10,10,10,10)
> vetor2
[1] 10 10 10 10
> vetor1 + vetor2
[1] 11 12 13 14
```


32

Vetores: Operações.

- Com vetores de tamanhos diferentes

```
> vetor1 <- 1:4
> vetor2 <- 1:8
> vetor1
[1] 1 2 3 4
> vetor2
[1] 1 2 3 4 5 6 7 8
> vetor1 + vetor2
[1]  2  4  6  8  6  8 10 12
```

- Neste caso o `vetor1` foi repetido (deve-se ter tamanhos múltiplos)

```
Vetor1:     1  2  3  4  1  2  3  4
Vetor2:     1  2  3  4  5  6  7  8
+           ----------------------
Resultado: 2  4  6  8  6  8 10 12
```


Vetores: Funções

- Obter o vetor reverso : `rev`
- Para obter a cauda do vetor: `tail` (default = 6)
- Para obter a cabeça do vetor : `head` (default = 6)

```
> x <- 20:30
> x
 [1] 20 21 22 23 24 25 26 27 28 29 30
> rev(x)
 [1] 30 29 28 27 26 25 24 23 22 21 20
> tail(x)
[1] 25 26 27 28 29 30
> head(x)
[1] 20 21 22 23 24 25
```

Vetores: Funções

- Verifica se algum dos elementos de um vetor tem uma condição : `any`
- Verifica se todos os elementos de um vetor tem uma condição : `all`

```
> x <- 10:15
> x
[1] 10 11 12 13 14 15
> any(x > 12)
[1] TRUE
> any(x < 5)
[1] FALSE
> all(x >= 15)
[1] FALSE
> all(x <= 100)
[1] TRUE
```


35

Vetores: Funções

- Ordenação : `sort` e `order`
  - `sort` : Retorna um vetor com os **valores ordenados**
  - `order` : Retorna um vetor com os **índices dos valores ordenados**
  - Parâmetro `decreasing` (booleano) indica se é decrescente ou não
- Ordenação : `sort`

```
> x <- c(15, 10, 19, 8)
> x
[1] 15 10 19  8
> sort(x)
[1]  8 10 15 19
> order(x)
[1] 4 2 1 3
> order(x, decreasing=TRUE)
[1] 3 1 2 4
```


36

Vetores: Funções.

Função	Descrição
`length(x)`	Número de elementos no vetor x
`sum(x)`	Soma dos elementos do vetor x
`prod(x)`	Produto dos elementos do vetor x
`max(x)`	Maior valor do vetor x
`min(x)`	Menor valor do vetor x
`range(x)`	Retorna o menor e o maior elemento do vetor x
`mean(x)`	Calcula a média dos valores em x

37

Exercícios.

1. Execute os exemplos apresentados nos slides

2. Dadas as leituras mensais em um medidor de consumo de luz

Jan	Fev	Mar	Abr	Mai	Jun	Jul	Ago	Set	Out	Nov	Dez
9839	10149	10486	10746	11264	11684	12082	12599	13004	13350	13717	14052

a) Crie um vetor com todas as leituras.
b) Calcule a média das leituras no período
c) Calcule o máximo e o mínimo das leituras no período
d) Ordene as medidas de forma crescente e decrescente

38