

Exercícios R-3

1 Execute os exemplos apresentados nos slides

a) Pode-se criar um vetor vazio com vector()

```
In [5]: x<-vector()
```

```
In [6]: print(x)
```

```
logical(0)
```

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

```
In [8]: x <- vector("integer", length=5)
```

```
In [9]: print(x)
```

```
[1] 0 0 0 0 0
```

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

```
In [10]: x<-integer(5)
```

```
In [11]: print(x)
```

```
[1] 0 0 0 0 0
```

d) Outro exemplo

```
In [14]: x<-character(5)
```

```
In [15]: print(x)
```

```
[1] "" "" "" "" ""
```

e) Pode-se criar um vetor indicando seu conteúdo usando a função:

```
In [16]: c()
```

```
NULL
```

exemplo:

```
In [18]: vector1 <- c(1,2,3,4)  
vector1
```

```
1.1
```

```
2.2
```

```
3.3
```

```
4.4
```

```
In [19]: class(vector1)
```

'numeric'

```
In [20]: typeof(vector1)
```

'double'

f) Se quiser forçar o uso de inteiros acrescente L

```
In [23]: vector1 <- c(1L,2L,3L,4L)
vector1
```

1. 1

2. 2

3. 3

4. 4

```
In [24]: class(vector1)
```

'integer'

```
In [25]: typeof(vector1)
```

'integer'

g) Outro exemplo de uso da função c()

```
In [27]: vector2 <- c("a","b","c","d")
vector2
```

1. 'a'

2. 'b'

3. 'c'

4. 'd'

h) 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

• Usando operador :

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

```
In [28]: vet <- 1:10
vet
```

1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7
8. 8
9. 9
10. 10

```
In [29]: vet <- 7:1  
vet
```

1. 7
2. 6
3. 5
4. 4
5. 3
6. 2
7. 1

```
In [30]: vet <- -3:1  
vet
```

1. -3
2. -2
3. -1
4. 0
5. 1

i) 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

j) Gera uma sequência de 1 a 10

```
In [33]: vet <- seq(10)
vet
```

```
1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7
8. 8
9. 9
10. 10
```

k) Gera uma sequência de 1 a -5

```
In [36]: vet <- seq(-5)
vet
```

```
1. 1
2. 0
3. -1
4. -2
5. -3
6. -4
7. -5
```

l) Gera uma sequência de 2 a 7

```
In [37]: vet <- seq(2,7)
vet
```

```
1. 2
2. 3
3. 4
4. 5
5. 6
6. 7
```

m) Sequências geradas com valores decimais

```
In [38]: seq(1.5,3.5,0.5)
```

1. 1.5
2. 2
3. 2.5
4. 3
5. 3.5

```
In [39]: seq(4.5, 2.0, -0.5)
```

1. 4.5
2. 4
3. 3.5
4. 3
5. 2.5
6. 2

n) Gera uma sequência de 1 a 10, de 2 em 2

```
In [40]: vet <- seq(1, 10, 2)
vet
```

1. 1
2. 3
3. 5
4. 7
5. 9

o) Gera uma sequência de 1 a 4, de 0.5 em 0.5

```
In [41]: vet <- seq(1, 4, 0.5)
vet
```

1. 1
2. 1.5
3. 2
4. 2.5
5. 3
6. 3.5
7. 4

p) Gera uma sequência de 1 a 4, contendo 10 elementos

```
In [42]: vet <- seq(1, 4, length.out=10)
vet
```

```
1. 1
2. 1.3333333333333333
3. 1.6666666666666667
4. 2
5. 2.3333333333333333
6. 2.6666666666666667
7. 3
8. 3.3333333333333333
9. 3.6666666666666667
10. 4
```

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

```
In [43]: vetor <- c(4,1,19,3,15)
vet <- seq(1, 10, along.with=vetor)
vet
```

```
1. 1
2. 3.25
3. 5.5
4. 7.75
5. 10
```

r) 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

```
In [44]: seq_along(vetor)
```

```
1. 1
2. 2
3. 3
4. 4
5. 5
```

s) 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

Repete 10, 3 vezes:

```
In [45]: rep(10,3)
```

1. 10

2. 10

3. 10

t) Repete o vetor 4 vezes

```
In [46]: rep(5:8, 4)
```

1. 5

2. 6

3. 7

4. 8

5. 5

6. 6

7. 7

8. 8

9. 5

10. 6

11. 7

12. 8

13. 5

14. 6

15. 7

16. 8

u) Repete o vetor até que o tamanho do resultado seja 7

```
In [47]: rep(5:6, length.out=7)
```

1. 5

2. 6

3. 5

4. 6

5. 5

6. 6

7. 5

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

```
In [48]: rep(c(10,20), times = c(2,4))
```

```
1. 10
2. 10
3. 20
4. 20
5. 20
6. 20
```

```
In [49]: rep(c(1,2), times = c(5,3))
```

```
1. 1
2. 1
3. 1
4. 1
5. 1
6. 2
7. 2
8. 2
```

w) Criar vetor com elementos repetidos: rep

```
In [50]: rep(10,5)
```

```
1. 10
2. 10
3. 10
4. 10
5. 10
```

```
In [51]: rep(c(1,2),3)
```

```
1. 1
2. 2
3. 1
4. 2
5. 1
6. 2
```

```
In [52]: rep(c(1,2), each=3)
```

```
1. 1
2. 1
3. 1
4. 2
5. 2
6. 2
```


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

$[logical] \Rightarrow [integer] \Rightarrow [double] \Rightarrow [complex] \Rightarrow [character]$

exemplo:

```
In [53]: vet <- c(TRUE, 1L, 10.2, "razer")
vet
```

```
1. 'TRUE'
2. '1'
3. '10.2'
4. 'razer'
```

Na coerção de logical para um tipo numérico:

- TRUE → 1
- FALSE → 0

y) As funções as.xxxx() convertem todos os elementos do vetor

Exemplo:

```
In [54]: x <- 1:3
class(x)
```

```
'integer'
```

```
In [55]: as.character(x)
```

```
1. '1'
2. '2'
3. '3'
```

z) Indexação: Acesso aos dados de um vetor

- Inicia em 1
- Operador []

Exemplo:

```
In [56]: vetor3 <- 1:10
vetor3
```

1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7
8. 8
9. 9
10. 10

```
In [57]: vetor3[5]
```

5

```
In [58]: vetor3[5] = 777
```

```
In [59]: vetor3
```

1. 1
2. 2
3. 3
4. 4
5. 777
6. 6
7. 7
8. 8
9. 9
10. 10

aa) Pode-se retornar mais de um elemento

- Usa-se `c()` dentro da indexação do vetor

Exemplo

```
In [60]: vetor <- 15:24  
vetor
```

1. 15
2. 16
3. 17
4. 18
5. 19
6. 20
7. 21
8. 22
9. 23
10. 24

```
In [61]: vetor[c(2,5)]
```

1. 16
2. 19

ab) Para adicionar um elemento ou vetor: append

```
In [62]: x <- 1:3
```

```
In [63]: y <- 4:6
```

```
In [64]: x
```

1. 1
2. 2
3. 3

```
In [65]: y
```

1. 4
2. 5
3. 6

```
In [66]: append(x,y)
```

1. 1
2. 2
3. 3
4. 4
5. 5
6. 6

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

```
In [67]: x <- 20:30
```

```
In [68]: x
```

1. 20
2. 21
3. 22
4. 23
5. 24
6. 25
7. 26
8. 27
9. 28
10. 29
11. 30

```
In [69]: x[3]
```

22

```
In [70]: x[-3]
```

1. 20
2. 21
3. 23
4. 24
5. 25
6. 26
7. 27
8. 28
9. 29
10. 30

ad) Para remover os elementos em uma faixa de valores de índices

```
In [71]: x <- 50:70  
x
```

1. 50
2. 51
3. 52
4. 53
5. 54
6. 55
7. 56
8. 57
9. 58
10. 59
11. 60
12. 61
13. 62
14. 63
15. 64
16. 65
17. 66
18. 67
19. 68
20. 69
21. 70

```
In [72]: x[-(5:10)]
```

1. 50
2. 51
3. 52
4. 53
5. 60
6. 61
7. 62
8. 63
9. 64
10. 65
11. 66
12. 67
13. 68
14. 69
15. 70

ae) Para remover um elemento pelo seu valor

```
In [74]: x <- 50:70
```

```
In [75]: x
```

1. 50
2. 51
3. 52
4. 53
5. 54
6. 55
7. 56
8. 57
9. 58
10. 59
11. 60
12. 61
13. 62
14. 63
15. 64
16. 65
17. 66
18. 67
19. 68
20. 69
21. 70

In [76]: `x[x!=51]`

1. 50
2. 52
3. 53
4. 54
5. 55
6. 56
7. 57
8. 58
9. 59
10. 60
11. 61
12. 62
13. 63
14. 64
15. 65
16. 66
17. 67
18. 68
19. 69
20. 70

af) Para remover elementos por uma lista de valores

```
In [77]: x <- 50:70  
x
```

```
1. 50  
2. 51  
3. 52  
4. 53  
5. 54  
6. 55  
7. 56  
8. 57  
9. 58  
10. 59  
11. 60  
12. 61  
13. 62  
14. 63  
15. 64  
16. 65  
17. 66  
18. 67  
19. 68  
20. 69  
21. 70
```

```
In [79]: x[!x %in% c(53,55,66)]
```

```
1. 50  
2. 51  
3. 52  
4. 54  
5. 56  
6. 57  
7. 58  
8. 59  
9. 60  
10. 61  
11. 62  
12. 63  
13. 64  
14. 65  
15. 67  
16. 68  
17. 69  
18. 70
```

ag) Obter Tamanho do vetor : length(vet)

Obter o último elemento : vet[length(vet)]

```
In [80]: x <- 20:30  
x
```

```
1. 20  
2. 21  
3. 22  
4. 23  
5. 24  
6. 25  
7. 26  
8. 27  
9. 28  
10. 29  
11. 30
```

```
In [81]: length(x)
```

```
11
```

```
In [82]: x[length(x)]
```

```
30
```

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

Exemplo:

```
In [83]: vetor1 <- c(1,2,3,4)  
vetor1-1
```

```
1. 0  
2. 1  
3. 2  
4. 3
```

```
In [84]: x <- vetor1 *10
```

```
In [85]: x
```

```
1. 10  
2. 20  
3. 30  
4. 40
```

```
In [86]: vetor1/2
```


1. 0.5
2. 1
3. 1.5
4. 2

a) Também consegue-se fazer operações entre vetores

Exemplo:

```
In [87]: vetor1 * vetor1
```

1. 1
2. 4
3. 9
4. 16

```
In [88]: vetor2 <- c(10,10,10,10)
vetor2
```

1. 10
2. 10
3. 10
4. 10

```
In [89]: vetor1 + vetor2
```

1. 11
2. 12
3. 13
4. 14

aj) Com vetores de tamanhos diferentes

```
In [90]: vetor1 <- 1:4
vetor2 <- 1:8
vetor1
```

1. 1
2. 2
3. 3
4. 4

```
In [91]: vetor2
```

1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7
8. 8

In [92]: `vetor1 + vetor2`

1. 2
2. 4
3. 6
4. 8
5. 6
6. 8
7. 10
8. 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

Obter o vetor reverso : `rev`

Para obter a cauda do vetor: `tail` (default = 6)

Para obter a cabeça do vetor : `head` (default = 6)

In [93]: `x <- 20:30`
`x`

1. 20
2. 21
3. 22
4. 23
5. 24
6. 25
7. 26
8. 27
9. 28
10. 29
11. 30

```
In [94]: rev(x)
```

```
1. 30
2. 29
3. 28
4. 27
5. 26
6. 25
7. 24
8. 23
9. 22
10. 21
11. 20
```

```
In [95]: tail(x)
```

```
1. 25
2. 26
3. 27
4. 28
5. 29
6. 30
```

```
In [96]: head(x)
```

```
1. 20
2. 21
3. 22
4. 23
5. 24
6. 25
```

ak) Verifica se algum dos elementos de um vetor tem uma condição : any

Verifica se todos os elementos de um vetor tem uma condição : all

```
In [97]: x <- 10:15
x
```

```
1. 10
2. 11
3. 12
4. 13
5. 14
6. 15
```

```
In [98]: any(x>12)
```

TRUE

In [99]: `any(x<5)`

FALSE

In [100... `all(x>=15)`

FALSE

In [102... `all(x<=100)`

TRUE

al) 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

In [103... `x <- c(15,10,19,8)`

In [104... `x`

1. 15
2. 10
3. 19
4. 8

In [105... `sort(x)`

1. 8
2. 10
3. 15
4. 19

In [106... `order(x)`

1. 4
2. 2
3. 1
4. 3

In [107... `order(x, decreasing=TRUE)`

- 1.3
- 2.1
- 3.2
- 4.4

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

```
In [110...] dados <- read.csv("dados1.csv", sep=";")
```

```
In [111...] dados
```

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

```
In [112...] vetor <- c(9839,10149,10486,10746,11264,11684,12082,12599,13004,13350,13717,14052)
```

```
In [113...] vetor
```

1. 9839
2. 10149
3. 10486
4. 10746
5. 11264
6. 11684
7. 12082
8. 12599
9. 13004
10. 13350
11. 13717
12. 14052

b) Calcule a média das leituras no período

```
In [114...] mean(vetor)
```

11914.3333333333

c) Calcule o máximo e o mínimo das leituras no período

```
In [115...] range(vetor)
```

1. 9839
2. 14052

d) Ordene as medidas de forma crescente e decrescente

In [117... `sort(vetor)`

```
1. 9839
2. 10149
3. 10486
4. 10746
5. 11264
6. 11684
7. 12082
8. 12599
9. 13004
10. 13350
11. 13717
12. 14052
```

In [118... `rev(sort(vetor))`

```
1. 14052
2. 13717
3. 13350
4. 13004
5. 12599
6. 12082
7. 11684
8. 11264
9. 10746
10. 10486
11. 10149
12. 9839
```

In []: