| Por favor, selecione um curso ou digite 0 para sair do swirl.

1: Aprenda R no R

2: Leve-me para o repositório de cursos do swirl!

Selection: 1

| Por favor, selecione uma lição ou digite 0 para retornar ao menu do curso.

1: Comandos basicos do R 2: Logicos 3: Sequencias numericas 4: Vetores

5: Valores ausentes 6: Data e Hora 7: Filtrando vetores 8: Matrizes e tabelas

9: Amostras e Simulacoes 10: Funcoes 11: Manipulacao com dplyr 12: Graficos Basicos

13: K Medias 14: Graficos Analiticos

Selection: 2

| Tentando carregar as dependências da lição...

| O pacote ‘openssl’ carregou corretamente!

| O pacote ‘jsonlite’ carregou corretamente!

| | 0%

| Esta lição deve ser uma breve introdução sobre valores booleanos e operações lógicas em R.

...

|== | 2%

| Existem dois valores lógicos em R, também chamados de valores booleanos. Eles são TRUE e FALSE. Em R você

| pode construir expressões lógicas que serão avaliadas como TRUE ou FALSE.

...

|==== | 4%

| Muitas das perguntas desta lição envolverão a avaliação de expressões lógicas. Se tiver dificuldades, pode

| ser útil abrir um segundo terminal R onde você pode experimentar algumas dessas expressões. Fica a seu

| critério.

...

|====== | 6%

| Criar expressões lógicas requer operadores lógicos. Você provavelmente está familiarizado com operadores

| aritméticos como `+`, `-`,`\*` e `/`. O primeiro operador lógico que vamos discutir é o operador de

| igualdade, representado por dois sinais de igual `==`. Use o operador de igualdade abaixo para descobrir se

| TRUE é igual a TRUE.

>

> TRUE == TRUE

[1] TRUE

| Excelente!

|======== | 8%

| Assim como a aritmética, as expressões lógicas podem ser agrupadas por parênteses para que a expressão

| inteira (TRUE == TRUE) == TRUE seja avaliada como TRUE.

...

|========== | 10%

| Para testar essa propriedade, tente avaliar (FALSE == TRUE) == FALSE.

> (FALSE == TRUE) == FALSE

[1] TRUE

| Sua dedicação é inspiradora!

|============ | 12%

| O operador de igualdade também pode ser usado para comparar números. Use `==` para ver se 6 é igual a 7.

> 6==7

[1] FALSE

| Você acertou!

|============== | 13%

| A expressão anterior é avaliada como FALSE porque 6 é menor que 7. Felizmente, existem operadores de

| desigualdade que nos permitem testar se um valor é menor ou maior que outro valor.

...

|================ | 15%

| O operador menor que '<' testa se o número no lado esquerdo do operador (chamado de operando à esquerda) é

| menor que o número no lado direito do operador (chamado operando à direita). Escreva uma expressão para

| testar se 6 é menor que 7.

> 6<7

[1] TRUE

| Bom trabalho!

|================= | 17%

| Existe também um operador menor que ou igual a <= que testa se o operando da esquerda é menor ou igual ao

| operando da direita. Escreva uma expressão para testar se 10 é menor ou igual a 10.

> 10<=10

[1] TRUE

| Continue assim e você chegará lá!

|=================== | 19%

| Tenha em mente que existem os operadores correspondentes maiores que `>` e maior que ou igual a `>=`.

...

|===================== | 21%

| Qual das seguintes opções é avaliada como FALSE?

1: 0 > -36

2: 7 == 7

3: 6 < 8

4: 9 >= 10

Selection: 4

| Você é muito bom, amig@!

|======================= | 23%

| Qual das seguintes avaliações é verdadeira?

1: 7 == 9

2: 57 < 8

3: -6 > -7

4: 9 >= 10

Selection: 3

| Excelente!

|========================= | 25%

| O próximo operador que discutiremos é o operador `não é igual` representado por `!=`. Não é igual testa se

| dois valores são diferentes, então TRUE != FALSE é avaliado como TRUE. Como o operador de igualdade, `!=`

| Também pode ser usado com números. Tente escrever uma expressão par ver se 5 não é igual a 7.

> 5!=7

[1] TRUE

| Na mosca! Bom trabalho!

|=========================== | 27%

| Para negar expressões booleanas, você pode usar o operador NOT. Um ponto de exclamação `!` Fará com que

| !TRUE (diga: não é verdade) avalie para FALSE e! FALSE (diga: não false) para avaliar como TRUE. Tente usar

| o operador NOT e o operador igual para encontrar o oposto de se 5 é igual a 7.

> NOT(5==7)

Error in NOT(5 == 7) : could not find function "NOT"

> !5==7

[1] TRUE

| Continue assim e você chegará lá!

|============================= | 29%

| Vamos dar um tempo para rever. O operador equals `==` testa se dois valores booleanos ou números são iguais,

| o operador não igual `! =` Testa se dois valores booleanos ou números são desiguais, e o operador NOT `!`

| nega expressões lógicas para que expressões TRUE se tornem Expressões FALSE e FALSE se tornam TRUE.

...

|=============================== | 31%

| Qual das seguintes opções é avaliada como FALSE?

1: 7 != 8

2: !FALSE

3: !(0 >= -1)

4: 9 < 10

Selection: 3

| Sua dedicação é inspiradora!

|================================= | 33%

| Em que você acha que a seguinte expressão será avaliada ?: (TRUE != FALSE) == !(6 == 7)

1: TRUE

2: %>%

3: FALSE

4: Pode haver verdade objetiva ao programar?

Selection: 1

| Você está em um bom ritmo!

|=================================== | 35%

| Em algum momento, você pode precisar examinar os relacionamentos entre várias expressões lógicas. É aqui que

| entra o operador AND e o operador OR.

...

|===================================== | 37%

| Vamos ver como o operador AND funciona. Existem dois operadores AND em R, `&` e `&&`. Ambos os operadores

| funcionam de forma semelhante, se os operandos direito e esquerdo de AND forem ambos TRUE a expressão

| inteira é TRUE, caso contrário, é FALSE. Por exemplo, TRUE & TRUE é avaliado como TRUE. Tente digitar FALSE

| & FALSE como é avaliado.

> FALSE & FALSE

[1] FALSE

| Excelente!

|======================================= | 38%

| Você pode usar o operador `&` para avaliar AND através de um vetor. A versão `&&` do AND só avalia o

| primeiro membro de um vetor. Vamos testar ambos para praticar. Digite a expressão TRUE & c(TRUE, FALSE,

| FALSE).

> TRUE & c(TRUE, FALSE, FALSE)

[1] TRUE FALSE FALSE

| Excelente trabalho!

|========================================= | 40%

| O que acontece nesse caso é que o operando da esquerda TRUE é reciclado em todos os elementos no vetor do

| operando da direita. Esta é a declaração equivalente a c(TRUE, TRUE, TRUE) & c(TRUE, FALSE, FALSE).

...

|=========================================== | 42%

| Agora vamos digitar a mesma expressão, exceto que usaremos o operador `&&`. Digite a expressão TRUE &&

| c(TRUE, FALSE, FALSE).

> TRUE && c(TRUE, FALSE, FALSE)

[1] TRUE

Warning message:

In TRUE && c(TRUE, FALSE, FALSE) :

'length(x) = 3 > 1' in coercion to 'logical(1)'

| Sua dedicação é inspiradora!

|============================================= | 44%

| Nesse caso, o operando esquerdo é avaliado apenas com o primeiro membro do operando direito (o vetor). O

| restante dos elementos no vetor não é avaliado de forma alguma nessa expressão.

...

|=============================================== | 46%

| O operador OR segue um conjunto similar de regras. A versão `|` do OR avalia OR através de um vetor inteiro,

| enquanto a versão || do OR apenas avalia o primeiro membro de um vetor.

...

|================================================= | 48%

| Uma expressão usando o operador OR será avaliada como TRUE se o operando esquerdo ou o operando da direita

| for TRUE. Se ambos forem TRUE, a expressão será avaliada como TRUE, no entanto, se nenhum for TRUE, a

| expressão será FALSE.

...

|================================================== | 50%

| Vamos testar a versão vetorizada do operador OR. Digite a expressão TRUE | c(TRUE, FALSE, FALSE).

> TRUE | c(TRUE, FALSE, FALSE)

[1] TRUE TRUE TRUE

| Bom trabalho!

|==================================================== | 52%

| Agora vamos experimentar a versão não vetorizada do operador OR. Digite a expressão TRUE || c(TRUE, FALSE,

| FALSE).

>

> TRUE || c(TRUE, FALSE, FALSE)

[1] TRUE

| Mantenha esse bom nível!

|====================================================== | 54%

| Os operadores lógicos podem ser encadeados como operadores aritméticos. As expressões: `6! = 10 && FALSE &&

| 1> = 2` ou `TRUE || 5 < 9.3 || FALSE` são até comuns de se ver.

...

|======================================================== | 56%

| Como você deve se lembrar, a aritmética tem uma ordem de operações e também expressões lógicas. Todos os

| operadores AND são avaliados antes dos operadores OR. Vamos ver um exemplo de um caso ambíguo. Tipo: 5 > 8

| || 6 != 8 && 4 > 3.9

>

> 5 > 8 || 6 != 8 && 4 > 3.9

[1] TRUE

| Continue assim e você chegará lá!

|========================================================== | 58%

| Vamos percorrer a ordem das operações no caso acima. Primeiro, os operandos esquerdo e direito do operador

| AND são avaliados. 6 não é igual a 8, 4 é maior que 3.9, portanto ambos os operandos são TRUE, então a

| expressão resultante TRUE && TRUE é avaliada como TRUE. Em seguida, o operando esquerdo do operador OR é

| avaliado: 5 não é maior que 8, portanto, a expressão inteira é reduzida para FALSE || TRUE. Como o operando

| direito dessa expressão é TRUE, toda a expressão é avaliada como TRUE.

...

|============================================================ | 60%

| Qual das seguintes expressões é avaliada como TRUE?

1: TRUE && FALSE || 9 >= 4 && 3 < 6

2: TRUE && 62 < 62 && 44 >= 44

3: FALSE || TRUE && FALSE

4: 99.99 > 100 || 45 < 7.3 || 4 != 4.0

Selection: 1

| Continue assim e você chegará lá!

|============================================================== | 62%

| Qual das seguintes expressões é avaliada como FALSE?

1: FALSE && 6 >= 6 || 7 >= 8 || 50 <= 49.5

2: FALSE || TRUE && 6 != 4 || 9 > 4

3: 6 >= -9 && !(6 > 7) && !(!TRUE)

4: (8 > 4) || 5 == 5.0 && 7.8 >= 7,79

Selection: 1

| Ótimo trabalho!

|================================================================ | 63%

| Agora que você está familiarizado com os operadores lógicos de R, você pode tirar proveito de algumas

| funções que o R fornece para lidar com expressões lógicas.

...

|================================================================== | 65%

| A função isTRUE() recebe um argumento. Se esse argumento for avaliado como TRUE, a função retornará TRUE.

| Caso contrário, a função retornará FALSE. Tente usar essa função digitando: isTRUE(6 > 4)

> isTRUE(6>4)

[1] TRUE

| Perseverança é a resposta.

|==================================================================== | 67%

| Qual das seguintes avaliações é verdadeira?

1: isTRUE(!TRUE)

2: istrue(NA)

3: !isTRUE(4 < 3)

4: !isTRUE(8 != 5)

5: istrue(3)

Selection: 3

| Excelente trabalho!

|====================================================================== | 69%

| A função identical() retornará TRUE se os dois objetos R passados como argumentos forem idênticos.

| Experimente a função identical() digitando: identical('twins', 'twins')

> identical('twins', 'twins')

[1] TRUE

| Sua dedicação é inspiradora!

|======================================================================== | 71%

| Qual das seguintes avaliações é verdadeira?

1: identical('olá', 'alô')

2: identical(7, 8)

3: identical(5 > 4, 3 < 3.1)

4: identical(4, 3.1)

Selection: 3

| Está correto!

|========================================================================== | 73%

| Você também deve estar ciente da função xor(), que leva dois argumentos. A função xor() significa OU

| exclusivo. Se um argumento for avaliado como TRUE e um argumento for avaliado como FALSE, essa função

| retornará TRUE, caso contrário, retornará FALSE. Experimente a função xor() digitando: xor(5 == 6, !FALSE)

> xor(5==6, !FALSE)

[1] TRUE

| Você está em um bom ritmo!

|============================================================================ | 75%

| 5 == 6 avalia como FALSE, !FALSE avalia como TRUE, então xor (FALSE, TRUE) é avaliado como TRUE. Por outro

| lado, se o primeiro argumento foi alterado para 5 == 5 e o segundo argumento não foi alterado, então ambos

| os argumentos teriam sido TRUE, então xor (TRUE, TRUE) teria avaliado como FALSE.

...

|============================================================================== | 77%

| Qual das seguintes opções é avaliada como FALSE?

1: xor(identical(xor, 'xor'), 7 == 7.0)

2: xor(!!TRUE, !!FALSE)

3: xor(4 >= 9, 8 != 8.0)

4: xor(!isTRUE(TRUE), 6 > -1)

Selection: 3

| Está correto!

|================================================================================ | 79%

| Para as próximas perguntas, precisaremos criar um vetor de inteiros chamados ints. Crie este vetor

| digitando: ints <- sample(10)

> ints <- sample(10)

| Você está indo muito bem!

|================================================================================== | 81%

| Agora simplesmente exiba o conteúdo de ints.

> ints

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

| Perseverança é a resposta.

|==================================================================================== | 83%

| O vetor `ints` é uma amostragem aleatória de números inteiros de 1 a 10 sem substituição. Digamos que

| quiséssemos fazer algumas perguntas lógicas sobre o conteúdo de ints. Se digitarmos ints > 5, obteremos um

| vetor lógico correspondente a cada elemento de ints for maior que 5. Tente digitar: ints > 5

> ints > 5

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

| Excelente trabalho!

|===================================================================================== | 85%

| Podemos usar o vetor lógico resultante para fazer outras perguntas sobre o ints. A função which() usa um

| vetor lógico como argumento e retorna os índices do vetor que são TRUE. Por exemplo, which(c(TRUE, FALSE,

| TRUE)) retornaria o vetor c(1, 3).

...

|======================================================================================= | 87%

| Use a função which() para encontrar os índices de ints maiores que 7.

> which(ints>7)

[1] 4 6 10

| Maravilha!

|========================================================================================= | 88%

| Qual dos seguintes comandos produziria os índices dos elementos em ints menores ou iguais a 2?

1: which(ints <= 2)

2: ints <= 2

3: ints < 2

4: which(ints <2)

Selection: 1

| Toda a prática está rendendo frutos!

|=========================================================================================== | 90%

| Como a função which(), as funções any() e all() usam vetores lógicos como argumento. A função any()

| retornará TRUE se um ou mais dos elementos no vetor lógico forem TRUE. A função all () retornará TRUE se

| cada elemento no vetor lógico for TRUE.

...

|============================================================================================= | 92%

| Use a função any() para ver se algum dos elementos de ints é menor que zero.

> any(ints>0)

[1] TRUE

| Essa não é a resposta esperada, mas tente novamente. Ou digite info() para mais opções.

| Use a função any() no vetor lógico produzido por: `ints <0`

> any(ints<0)

[1] FALSE

| Você acertou!

|=============================================================================================== | 94%

| Use a função all() para ver se todos os elementos de ints são maiores que zero.

> all(>0)

Error: unexpected '>' in "all(>"

> all(ints>0)

[1] TRUE

| Você está indo muito bem!

|================================================================================================= | 96%

| Qual das seguintes avaliações é verdadeira?

1: any(ints == 10)

2: all(ints == 10)

3: any(ints == 2.5)

4: all(c(TRUE, FALSE, TRUE))

Selection: 1

| Na mosca! Bom trabalho!

|=================================================================================================== | 98%

| Isso é tudo para esta introdução à lógica em R. Se você realmente quer ver o que você pode fazer com a

| lógica, confira a lição de fluxo de controle!

...

|=====================================================================================================| 100%

| Gostaria de informar ao professor sobre a conclusão desta lição

1: Não

2: Sim