| As funções são fundamentais em qualquer linguagem de programação, inclusive na linguagem R. São pequenos pedaços de código reutilizável que podem ser tratados como qualquer outro objeto R.

...

|==== | 2%

| Se você já fez qualquer outra lição deste curso, provavelmente já usou algumas funções criadas por outras pessoas. As funções são geralmente caracterizadas pelo nome da função seguido por parênteses.

...

|======== | 4%

| Vamos tentar usar algumas funções básicas apenas por diversão. A função Sys.Date() retorna uma string representando a data de hoje. Digite Sys.Date() e veja o que acontece.

warning messages from top-level task callback 'mini'

Warning messages:

1: In doTryCatch(return(expr), name, parentenv, handler) :

display list redraw incomplete

2: In doTryCatch(return(expr), name, parentenv, handler) :

invalid graphics state

3: In doTryCatch(return(expr), name, parentenv, handler) :

invalid graphics state

> sys.Date()

Error in sys.Date() : could not find function "sys.Date"

> Sys.Date()

[1] "2022-07-23"

| Você está em um bom ritmo!

|============ | 6%

| A maioria das funções em R retorna um valor. Funções como Sys.Date() retornam um valor com base no ambiente do seu computador, enquanto outras funções manipulam dados de entrada para calcular um valor de saída (o valor de

| retorno).

...

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

| A função mean() usa um vetor de números como entrada e retorna a média de todos os números do vetor. Entradas para funções são freqüentemente chamadas de argumentos ou parâmetros. Os argumentos que você quer passar para uma

| função vão dentro dos parênteses da função. Tente passar o argumento c(2, 4, 5) para a função mean().

> mean(c(2,4,5))

[1] 3.666667

| Todo o trabalho duro está dando resultado!

|==================== | 9%

| Funções geralmente enxergam os parâmetros como variáveis com as quais a função pode trabalhar. Por exemplo, a função mean() usa um vetor como argumento, como no caso mean(c(2,6,8)). A função mean(), em seguida, soma todos os

| números do vetor e divide essa soma pelo comprimento do vetor.

...

|========================= | 11%

| Antes de fazermos nossa primeira função, preciso falar uma coisa importante sobre a estrutura de uma função no R. No console, quando você digita uma expressão sem atribuir a uma variável o R exibe o valor da expressão,

| certo? Algo similar acontece com as funções. Em uma função, se a última expressão 'executada' não for atribuída a uma variável se tornará o valor de retorno da função. Também é possível definir o retorno com a instrução

| return() como nas linguagens tradicionais. Você perceberá isso ao longo desta lição, mas achei interessante fazer esta introdução primeiro.

...

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

| Agora, vamos ao que interessa ...

...

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

| Na próxima questão, você será solicitado a modificar um script que aparecerá na sua tela. Quando terminar de modificar o script, salve suas alterações e digite submit() para que eu possa avaliar magicamente teu script!

| Haverá alguns comentários com instruções no script que será aberto, portanto, leia-os!

...

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

| Queremos que esta função receba um argumento x, e retorne x sem modificá-lo. Exclua o comentário para que x seja retornado sem qualquer modificação. Certifique-se de salvar seu script antes de digitar submit().

> submit()

| Lendo seu script...

| Essa é a resposta que eu esperava.

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

| Agora que você criou sua primeira função, vamos testá-la! Digite: boring\_function("Minha primeira função!"). Se a sua função funcionar, ela deve retornar a string: "Minha primeira função!"

> boring\_function('Minha primeira função!')

[1] "Minha primeira função!"

| Mantenha esse bom nível!

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

| Parabéns por escrever sua primeira função. Ao escrever funções, você pode começar a entender melhor como o R funciona. Como John Chambers, o criador de R disse uma vez: Para entender os cálculos em R, duas coisas são úteis:

| 1. Tudo o que existe é um objeto. 2. Tudo o que acontece é uma chamada de função.

...

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

| Se você quiser ver o código-fonte de qualquer função, basta digitar o nome da função sem argumentos ou parênteses. Vamos tentar isso com a função que você acabou de criar. Digite: boring\_function para visualizar seu

| código-fonte.

> boring\_function

function(x) {

x

}

<bytecode: 0x000001d99764b198>

| Maravilha!

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

| Hora de fazer uma função mais útil! Vamos replicar a funcionalidade da função mean() criando uma função chamada: my\_mean(). Lembre-se de que, para calcular a média de todos os números em um vetor, você encontra a soma de

| todos os números no vetor e depois divide essa soma pela quantidade de números no vetor.

...

|========================================================= | 26%

| Certifique-se de salvar seu script antes de digitar submit().

> submit()

| Lendo seu script...

| Ótimo trabalho!

|============================================================= | 28%

| Agora teste sua função my\_mean() encontrando a média do vetor c(4, 5, 10).

> my\_mean(4,5,10)

Error in my\_mean(4, 5, 10) : unused arguments (5, 10)

> my\_mean(c(4,5,10))

| Você é muito bom, amig@!

|================================================================== | 30%

| Agora vamos escrever uma função com argumentos que possuem valor padrão. Isso pode ser útil se você acha que alguém que usa sua função irá definir um determinado argumento para o mesmo valor na maioria das vezes.

...

|====================================================================== | 32%

| Certifique-se de salvar seu script antes de digitar submit(). Só assim eu consigo avaliar magicamente a sua função.

> submit()

| Lendo seu script...

| Você é muito bom, amig@!

|========================================================================== | 34%

| Vamos fazer alguns testes da função remainder que você criou. Execute remainder(5) e veja o que acontece.

> remainder(5)

| Todo o trabalho duro está dando resultado!

|============================================================================== | 36%

| Vamos analisar por um momento o que acabou de acontecer. Você forneceu um argumento para a função e o R entendeu que este argumento era o parâmetro "num", já que "num" é o primeiro parâmetro. O valor padrão para "divisor" é

| 2, então a função usou o valor padrão, já que você não forneceu um valor específico.

...

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

| Agora vamos testar a função remainder fornecendo dois argumentos. Digite: remainder(11, 5) e vamos ver o que acontece.

> remainder(11,5)

| Você é muito bom, amig@!

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

| Mais uma vez, os argumentos foram definidos corretamente.

...

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

| Você também pode especificar explicitamente parâmetros ao chamar uma função. Quando você define explicitamente valores de argumentos por nome, a ordenação dos argumentos perde a importância. Você pode tentar isso digitando:

| remainder(divisor = 11, num = 5)

> remainder(divisor=11, num=5)

| Você é muito bom, amig@!

|============================================================================================== | 43%

| Como você pode ver, há uma diferença significativa entre o remainder(11, 5) e remainder(divisor = 11, num = 5)! O primeiro exemplo é equivalente a remainder(num = 11, divisor = 5), o segundo é o oposto.

...

|================================================================================================== | 45%

| O R também pode encontrar parcialmente o nome dos parâmetros. Tente digitar remainder(4, div = 2) para ver esse recurso em ação.

> remainder(4, div=2)

| Sua dedicação é inspiradora!

|====================================================================================================== | 47%

| Uma dica de quem já pastou com isso: em geral, você quer tornar seu código tão fácil de entender quanto possível. Alternar as ordens de argumentos especificando seus nomes ou usando apenas nomes parciais de argumentos pode

| ser confuso. É bom você conhecer estes recursos para o caso de você ler códigos feitos por outras pessoas, mas evite fazer códigos assim!

...

|========================================================================================================== | 49%

| Com toda essa conversa sobre argumentos, você pode estar se perguntando se existe uma maneira de ver os argumentos de uma função sem precisar abrir a documentação. Siiiiiim! Você pode usar a função args()! Digite:

| args(remainder) para examinar os argumentos da função remainder.

> args(remainder)

function (num, divisor = 2)

NULL

| Está correto!

|=============================================================================================================== | 51%

| Você pode não perceber, mas eu enganei você para fazer algo muito interessante! args() é uma função, remainder() é uma função, mas remainder era um argumento para args(). Sim, é verdade: você pode passar funções como

| argumentos! Este é um conceito muito poderoso. Vamos escrever um script para ver como funciona.

...

|=================================================================================================================== | 53%

| Certifique-se de salvar seu script antes de digitar submit().

> submit()

| Lendo seu script...

| Toda a prática está rendendo frutos!

|======================================================================================================================= | 55%

| Vamos pegar sua nova função evaluate() e testar! Chame evaluate para encontrar o desvio padrão do vetor c(1.4, 3.6, 7.9, 8.8).

> evaluate(sd(c(1.4,3.6,7.9,8.8)))

Error in func(dat) : could not find function "func"

> evaluate(sd, c(1.4,3.6,7.9,8.8))

| Sua dedicação é inspiradora!

|=========================================================================================================================== | 57%

| A ideia de passar funções como parâmetros para outras funções é um conceito importante e fundamental em programação.

...

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

| Você pode se surpreender ao saber que pode passar uma função como um argumento sem primeiro definir a função passada. Funções que não são nomeadas são apropriadamente conhecidas como funções anônimas, ou ainda pelo apelido

| "lâmbda".

...

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

| Vamos usar a função evaluate que você criou para explorar como funcionam as funções anônimas. Para o primeiro argumento da função evaluate, vamos escrever uma pequena função que se encaixa em uma linha. No segundo argumento,

| passaremos alguns dados que serão usados pela função anônima.

...

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

| Digite o seguinte comando e depois discutiremos como ele funciona: evaluate(function (x) {x + 1}, 6)

> evaluate(function(x) {x+1}, 6)

| Mantenha esse bom nível!

|=========================================================================================================================================== | 64%

| O primeiro argumento é uma pequena função anônima. Esta função pega um argumento `x` e retorna `x + 1`. Nós passamos o número 6 para essa função, então a expressão inteira retornou 7.

...

|=============================================================================================================================================== | 66%

| Tente usar a sua função evaluate() junto com uma função anônima para retornar o primeiro elemento do vetor c(8, 4, 0). Para isso você deve se lembrar da aula de filtrar vetores. Sua função anônima deve ter apenas um

| argumento que deve ser uma variável `x`.

> evaluate(function(x) {x[1]}, c(1,2,3))

| Não exatamente, mas você está aprendendo! Tente novamente. Ou digite info() para mais opções.

| Você pode precisar se lembrar como fitlrar vetore. Lembre-se que sua função anônima deve ter apenas um argumento, e esse argumento deve ser chamado `x`. Como esta é difícil vou dar uma super dica ... Você deve chamar a

| função evaluate() passando o vetor c(8, 4, 0) no segundo parâmetro e uma função como esta "function(x){x[1]}" no primeiro

> evaluate(function(x) {x[1]}, c(8,4,0))

| Excelente trabalho!

|=================================================================================================================================================== | 68%

| Acredite, mesmo para programadores calejados as funções anônimas podem ser difíceis de entender.

...

|======================================================================================================================================================= | 70%

| Agora tente usar evaluate() junto com uma função anônima para retornar o ÚLTIMO elemento do vetor c(8, 4, 0). Sua função anônima deve ter apenas um argumento que deve ser uma variável `x`. Veja que é muito parecido ao

| exercício anterior, mas ao invés do primeiro elemento eu espero o último.

> evaluate(function(x) {x[-1]}, c(8,4,0))

| Toda a prática está rendendo frutos!

|============================================================================================================================================================ | 72%

| Para o resto da lição (e do curso), vamos usar bastante a função paste(). Digite ?paste para que possamos dar uma olhada na documentação para a função.

> ?paste

| Excelente trabalho!

|================================================================================================================================================================ | 74%

| Como você pode ver, o primeiro argumento de paste () é `...`, que é chamado de reticências, elipse ou simplesmente 3-pontos ou dot-dot-dot. As reticências permitem que um número indefinido de argumentos seja passado para uma

| função. No caso de paste(), qualquer número de strings pode ser passado como argumentos e paste() retornará todas as strings combinadas em uma única string.

...

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

| Apenas para ver na prática como o paste() funciona, digite paste("Programar", "é", "divertido!")

> paste('Programar', 'é', 'divertido')

[1] "Programar é divertido"

| Por pouco. Continue tentando. Ou digite info() para mais opções.

| Basta digitar: paste("Programar", "é", "divertido!")

> paste("Programar", "é", "divertido!")

[1] "Programar é divertido!"

| Continue assim e você chegará lá!

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

| Viu como funciona né ... agora é hora de escrever nossa própria versão modificada de paste().

...

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

| Certifique-se de salvar seu script antes de digitar submit().

> submit()

| Lendo seu script...

| Você acertou!

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

| Agora vamos testar sua função telegram(). Use sua nova função passando qualquer argumento que desejar!

> telegram('TOU CURTINDO R')

| Maravilha!

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

| Agora vamos para a próxima. Vou abrir um novo script na sua tela.

...

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

| Certifique-se de salvar seu script antes de digitar submit().

> submit()

| Lendo seu script...

| Ótimo trabalho!

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

| Hora de usar sua função mad\_libs. Certifique-se de nomear os argumentos place, adjective e noun para que a sua função funcione.

> mad\_libs(place, adjective, noun)

Error in mad\_libs(place, adjective, noun) : object 'place' not found

> mad\_libs(place='place', adjective='adjective', noun='noun')

[1] "Novidades diretas de place . Hoje, pessoas adjective tomaram as ruas em protesto contra o(a) novo(a) noun que foi instalado(a) na cidade."

| Na mosca! Bom trabalho!

|================================================================================================================================================================================================ | 89%

| Estamos chegando ao final desta lição, mas ainda há mais um conceito do qual você deve conhecer.

...

|===================================================================================================================================================================================================== | 91%

| Você está familiarizado com a adição, subtração, multiplicação e divisão de números em R. Para fazer isso, você usa os símbolos +, -, \* e /. Esses símbolos são chamados de operadores binários porque eles usam duas entradas,

| uma entrada da esquerda e uma entrada da direita.

...

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

| No R você pode definir seus próprios operadores binários. Vou abrir um arquivo com um roteiro, para você fazer isso.

...

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

| Certifique-se de salvar seu script antes de digitar submit().

> submit()

| Lendo seu script...

| Bom trabalho!

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

| Você fez seu próprio operador binário! Vamos testar isso. Digite as strings 'Eu', 'adoro', 'R!' usando seu novo operador binário.

> p('Eu', adoro, 'R!')

Error in p("Eu", adoro, "R!") : could not find function "p"

> "%p%"

[1] "%p%"

| Tente novamente. Acertar de primeira não tem graça mesmo! Ou digite info() para mais opções.

| Use %p% entre cada string. 'Eu', 'adoro', 'R!'

> 'Eu'"%p%"'adoro"%p%",'R!

Error: unexpected string constant in "'Eu'"%p%""

> 'Eu' %p% 'adoro' %p% 'R!'

[1] "Eu adoro R!"

| Você acertou!

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

| Chegamos ao final da nossa aula! Vá lá e escreva ótimas funções!

...

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

| Gostaria de informar ao professor sobre a conclusão desta lição

1: Não

2: Sim

Selection: 2

| Qual o código da sua turma? (exemplo FIAP-01IA)

24IA

| Qual seu código de aluno?

344154

| Qual seu nome?

Diego Cohen

| O que achou deste exercício?

a parte final ficou confusa...

[1] "Tentando submeter ao professor, tentativa 1 ... (max 5) ..."

[1] "saved"

#################################################################################################################################################################################################################################

Seu resultado foi salvo!

#################################################################################################################################################################################################################################