Datacamp_intermediate_R___sapply

dizhen

2019/4/2

sapply

- apply function over list or vector
- try to simplify list to array

```
cities <- c("New York", "Paris", "London", "Tokyo", "Rio de Janeiro", "Cape Town")
lapply(cities, nchar)
```

```
## [[1]]
## [1] 8
##
## [[2]]
## [1] 5
##
## [[3]]
## [1] 6
##
## [[4]]
## [1] 5
##
## [[5]]
## [1] 14
##
## [[6]]
## [1] 9
```

```
unlist(lapply(cities, nchar))
```

```
## [1]  8  5  6  5 14  9
```

```
sapply(cities, nchar)
```

```
##       New York          Paris         London          Tokyo Rio de Janeiro
##              8              5              6              5             14
##      Cape Town
##              9
```

```
sapply(cities, nchar, USE.NAMES = FALSE) # USE.NAMES is TRUE by default
```

```
## [1]  8  5  6  5 14  9
```

```r
first_and_last <- function(name) {
  name <- gsub(" ","", name)
  letters <- strsplit(name, split = "")[[1]]
  c(first = min(letters), last = max(letters))
}
first_and_last("New York")
```

```
## first  last
##   "e"   "Y"
```

```r
sapply(cities, first_and_last)
```

```
##       New York Paris London Tokyo Rio de Janeiro Cape Town
## first "e"      "a"   "d"    "k"   "a"            "a"
## last  "Y"      "s"   "o"    "y"   "R"            "w"
```

```r
unique_letters <- function(name) {
  name <- gsub(" ","", name)
  letters <- strsplit(name, split = "")[[1]]
  unique(letters)
}
unique_letters("London")
```

```
## [1] "L" "o" "n" "d"
```

```r
lapply(cities, unique_letters) # give a list
```

```
## [[1]]
## [1] "N" "e" "w" "Y" "o" "r" "k"
##
## [[2]]
## [1] "P" "a" "r" "i" "s"
##
## [[3]]
## [1] "L" "o" "n" "d"
##
## [[4]]
## [1] "T" "o" "k" "y"
##
## [[5]]
## [1] "R" "i" "o" "d" "e" "J" "a" "n" "r"
##
## [[6]]
## [1] "C" "a" "p" "e" "T" "o" "w" "n"
```

```r
sapply(cities, unique_letters) # give an array; sapply did not simplify. Can be dangerous
```

```
## $`New York`
## [1] "N" "e" "w" "Y" "o" "r" "k"
##
```

```
## $Paris
## [1] "P" "a" "r" "i" "s"
##
## $London
## [1] "L" "o" "n" "d"
##
## $Tokyo
## [1] "T" "o" "k" "y"
##
## $`Rio de Janeiro`
## [1] "R" "i" "o" "d" "e" "J" "a" "n" "r"
##
## $`Cape Town`
## [1] "C" "a" "p" "e" "T" "o" "w" "n"
```

Practice

How to use sapply

sapply(X, FUN, ...)

In the next couple of exercises, you'll be working with the variable temp, that contains temperature measurements for 7 days. temp is a list of length 7, where each element is a vector of length 5, representing 5 measurements on a given day. This variable has already been defined in the workspace: type str(temp) to see its structure.

1. Use lapply() to calculate the minimum (built-in function min()) of the temperature measurements for every day.
2. Do the same thing but this time with sapply(). See how the output differs.
3. Use lapply() to compute the the maximum (max()) temperature for each day.
4. Again, use sapply() to solve the same question and see how lapply() and sapply() differ.

```
# temp has already been defined in the workspace
temp <- list(c(3,7,9,6,-1),
             c(6,9,12,13,5),
             c(4,8,3,-1,-3),
             c(1,4,7,2,-2),
             c(5,7,9,4,2),
             c(-3,5,8,9,4),
             c(3,6,9,4,1))

# Use lapply() to find each day's minimum temperature
lapply(temp,min)
```

```
## [[1]]
## [1] -1
##
## [[2]]
## [1] 5
##
## [[3]]
```

```
## [1] -3
##
## [[4]]
## [1] -2
##
## [[5]]
## [1] 2
##
## [[6]]
## [1] -3
##
## [[7]]
## [1] 1
```

```
# Use sapply() to find each day's minimum temperature
sapply(temp,min)
```

```
## [1] -1  5 -3 -2  2 -3  1
```

```
# Use lapply() to find each day's maximum temperature
lapply(temp,max)
```

```
## [[1]]
## [1] 9
##
## [[2]]
## [1] 13
##
## [[3]]
## [1] 8
##
## [[4]]
## [1] 7
##
## [[5]]
## [1] 9
##
## [[6]]
## [1] 9
##
## [[7]]
## [1] 9
```

```
# Use sapply() to find each day's maximum temperature
sapply(temp,max)
```

```
## [1]  9 13  8  7  9  9  9
```

sapply with your own function

Like lapply(), sapply() allows you to use self-defined functions and apply them over a vector or a list:

sapply(X, FUN, . . .)

Here, FUN can be one of R's built-in functions, but it can also be a function you wrote. This self-written function can be defined before hand, or can be inserted directly as an anonymous function.

1. Finish the definition of extremes_avg(): it takes a vector of temperatures and calculates the average of the minimum and maximum temperatures of the vector.
2. Next, use this function inside sapply() to apply it over the vectors inside temp.
3. Use the same function over temp with lapply() and see how the outputs differ.

```
# temp is already defined in the workspace

# Finish function definition of extremes_avg
extremes_avg <- function(x) {
  ( min(x) + max(x) ) / 2
}

# Apply extremes_avg() over temp using sapply()
sapply(temp,extremes_avg)
```

```
## [1] 4.0 9.0 2.5 2.5 5.5 3.0 5.0
```

```
# Apply extremes_avg() over temp using lapply()
lapply(temp,extremes_avg)
```

```
## [[1]]
## [1] 4
##
## [[2]]
## [1] 9
##
## [[3]]
## [1] 2.5
##
## [[4]]
## [1] 2.5
##
## [[5]]
## [1] 5.5
##
## [[6]]
## [1] 3
##
## [[7]]
## [1] 5
```

sapply with function returning vector

In the previous exercises, you've seen how sapply() simplifies the list that lapply() would return by turning it into a vector. But what if the function you're applying over a list or a vector returns a vector of length greater than 1?

1. Finish the definition of the extremes() function. It takes a vector of numerical values and returns a vector containing the minimum and maximum values of a given vector, with the names "min" and "max", respectively.

2. Apply this function over the vector temp using sapply().

3. Finally, apply this function over the vector temp using lapply() as well.

```
# temp is already available in the workspace

# Create a function that returns min and max of a vector: extremes
extremes <- function(x) {
  c(min = min(x), max = max(x))
}

# Apply extremes() over temp with sapply()
sapply(temp,extremes)
```

```
##     [,1] [,2] [,3] [,4] [,5] [,6] [,7]
## min   -1    5   -3   -2    2   -3    1
## max    9   13    8    7    9    9    9
```

```
# Apply extremes() over temp with lapply()
lapply(temp,extremes)
```

```
## [[1]]
## min max
##  -1   9
##
## [[2]]
## min max
##   5  13
##
## [[3]]
## min max
##  -3   8
##
## [[4]]
## min max
##  -2   7
##
## [[5]]
## min max
##   2   9
##
## [[6]]
## min max
##  -3   9
##
## [[7]]
## min max
##   1   9
```

sapply can't simplify, now what?

It seems like we've hit the jackpot with sapply(). On all of the examples so far, sapply() was able to nicely simplify the rather bulky output of lapply(). But, as with life, there are things you can't simplify. How does sapply() react?

We already created a function, below_zero(), that takes a vector of numerical values and returns a vector that only contains the values that are strictly below zero.

1. Apply below_zero() over temp using sapply() and store the result in freezing_s.
2. Apply below_zero() over temp using lapply(). Save the resulting list in a variable freezing_l.
3. Compare freezing_s to freezing_l using the identical() function.

```
# temp is already prepared for you in the workspace

# Definition of below_zero()
below_zero <- function(x) {
  return(x[x < 0])
}

# Apply below_zero over temp using sapply(): freezing_s
freezing_s <- sapply(temp,below_zero)

# Apply below_zero over temp using lapply(): freezing_l
freezing_l <- lapply(temp,below_zero)

# Are freezing_s and freezing_l identical?
identical(freezing_s,freezing_l)
```

```
## [1] TRUE
```

sapply with functions that return NULL

You already have some apply tricks under your sleeve, but you're surely hungry for some more, aren't you? In this exercise, you'll see how sapply() reacts when it is used to apply a function that returns NULL over a vector or a list.

A function print_info(), that takes a vector and prints the average of this vector, has already been created for you. It uses the cat() function.

Notice here that, quite surprisingly, sapply() does not simplify the list of NULL's. That's because the 'vector-version' of a list of NULL's would simply be a NULL, which is no longer a vector with the same length as the input.

```
# temp is already available in the workspace

# Definition of print_info()
print_info <- function(x) {
  cat("The average temperature is", mean(x), "\n")
}

# Apply print_info() over temp using sapply()
sapply(temp,print_info)
```

```
## The average temperature is 4.8
## The average temperature is 9
## The average temperature is 2.2
## The average temperature is 2.4
## The average temperature is 5.4
## The average temperature is 4.6
## The average temperature is 4.6
```

```
## [[1]]
## NULL
##
## [[2]]
## NULL
##
## [[3]]
## NULL
##
## [[4]]
## NULL
##
## [[5]]
## NULL
##
## [[6]]
## NULL
##
## [[7]]
## NULL
```

```
# Apply print_info() over temp using lapply()
lapply(temp,print_info)
```

```
## The average temperature is 4.8
## The average temperature is 9
## The average temperature is 2.2
## The average temperature is 2.4
## The average temperature is 5.4
## The average temperature is 4.6
## The average temperature is 4.6
```

```
## [[1]]
## NULL
##
## [[2]]
## NULL
##
## [[3]]
## NULL
##
## [[4]]
## NULL
##
## [[5]]
## NULL
##
## [[6]]
## NULL
##
## [[7]]
## NULL
```

Reverse engineering sapply

```r
sapply(list(runif (10), runif (10)),
       function(x) c(min = min(x), mean = mean(x), max = max(x)))
```

```
##            [,1]      [,2]
## min  0.07329493 0.1768826
## mean 0.48389085 0.5579749
## max  0.95559086 0.8949913
```

Without going straight to the console to run the code, try to reason through which of the following statements are correct and why.

(1) sapply() can't simplify the result that lapply() would return, and thus returns a list of vectors.
(2) This code generates a matrix with 3 rows and 2 columns.
(3) The function that is used inside sapply() is anonymous.
(4) The resulting data structure does not contain any names.

Select the option that lists all correct statements. (2) and (3)