Datacamp_Intermediate_R___lapply

dizhen

2019/4/1

lapply

```
nyc <- list(pop = 8405837,
  boroughs = c("Manhattan", "Bronx", "Brooklyn","Queens", "Staten Island"),
  capital = FALSE)
lapply(nyc, class)
```

```
## $pop
## [1] "numeric"
##
## $boroughs
## [1] "character"
##
## $capital
## [1] "logical"
```

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

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

```
unlist(lapply(cities, nchar)) # change the list to a vector
```

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

```
oil_prices <- list(2.37, 2.49, 2.18, 2.22, 2.47, 2.32)
multiply <- function(x, factor) {
  x * factor
}
times3 <- lapply(oil_prices, multiply, factor = 3)
unlist(times3)
```

```
## [1] 7.11 7.47 6.54 6.66 7.41 6.96
```

Practice

Use lapply with a built-in R function

lapply(X, FUN, . . .)

1. Have a look at the strsplit() calls, that splits the strings in pioneers on the : sign. The result, split_math is a list of 4 character vectors: the first vector element represents the name, the second element the birth year.
2. Use lapply() to convert the character vectors in split_math to lowercase letters: apply tolower() on each of the elements in split_math. Assign the result, which is a list, to a new variable split_low.
3. Finally, inspect the contents of split_low with str().

```
# The vector pioneers has already been created for you
pioneers <- c("GAUSS:1777", "BAYES:1702", "PASCAL:1623", "PEARSON:1857")

# Split names from birth year
split_math <- strsplit(pioneers, split = ":")

# Convert to lowercase strings: split_low
split_low = lapply(split_math,tolower)

# Take a look at the structure of split_low
str(split_low)
```

```
## List of 4
##  $ : chr [1:2] "gauss" "1777"
##  $ : chr [1:2] "bayes" "1702"
##  $ : chr [1:2] "pascal" "1623"
##  $ : chr [1:2] "pearson" "1857"
```

Use lapply with your own function

1. Apply select_first() over the elements of split_low with lapply() and assign the result to a new variable names.
2. Next, write a function select_second() that does the exact same thing for the second element of an inputted vector.
3. Finally, apply the select_second() function over split_low and assign the output to the variable years.

```
# Code from previous exercise:
pioneers <- c("GAUSS:1777", "BAYES:1702", "PASCAL:1623", "PEARSON:1857")
split <- strsplit(pioneers, split = ":")
split_low <- lapply(split, tolower)

# Write function select_first()
select_first <- function(x) {
  x[1]
}

# Apply select_first() over split_low: names
names <- lapply(split_low,select_first)

# Write function select_second()
select_second <- function(x){
  x[2]
}

# Apply select_second() over split_low: years
years <- lapply(split_low,select_second)
```

lapply and anonymous functions

You can use so-called anonymous functions in R.This means that they aren't automatically bound to a name. When you create a function, you can use the assignment operator to give the function a name. It's perfectly possible, however, to not give the function a name. This is called an anonymous function:

```
# Named function
triple <- function(x) { 3 * x }

# Anonymous function with same implementation
function(x) { 3 * x }
```

```
## function(x) { 3 * x }
```

```
# Use anonymous function inside lapply()
lapply(list(1,2,3), function(x) { 3 * x })
```

```
## [[1]]
## [1] 3
##
## [[2]]
## [1] 6
##
## [[3]]
## [1] 9
```

1. Transform the first call of lapply() such that it uses an anonymous function that does the same thing.
2. In a similar fashion, convert the second call of lapply to use an anonymous version of the select_second() function.
3. Remove both the definitions of select_first() and select_second(), as they are no longer useful.

```
# split_low has been created for you
split_low
```

```
## [[1]]
## [1] "gauss" "1777"
##
## [[2]]
## [1] "bayes" "1702"
##
## [[3]]
## [1] "pascal" "1623"
##
## [[4]]
## [1] "pearson" "1857"
```

```
# Transform: use anonymous function inside lapply
names <- lapply(split_low, function(x){x[1]})

# Transform: use anonymous function inside lapply
years <- lapply(split_low, function(x){x[2]})
```

Use lapply with additional arguments

lapply() provides a way to handle functions that require more than one argument, such as the multiply() function:

```
multiply <- function(x, factor) {
  x * factor
}
lapply(list(1,2,3), multiply, factor = 3)
```

```
## [[1]]
## [1] 3
##
## [[2]]
## [1] 6
##
## [[3]]
## [1] 9
```

1. Use lapply() twice to call select_el() over all elements in split_low: once with the index equal to 1 and a second time with the index equal to 2. Assign the result to names and years, respectively.

```
# Definition of split_low
pioneers <- c("GAUSS:1777", "BAYES:1702", "PASCAL:1623", "PEARSON:1857")
split <- strsplit(pioneers, split = ":")
split_low <- lapply(split, tolower)

# Generic select function
select_el <- function(x, index) {
  x[index]
}
```

```
# Use lapply() twice on split_low: names and years
names <- lapply(split_low,select_el,index = 1)
years <- lapply(split_low,select_el,index = 2)
```

Apply functions that return NULL

In all of the previous exercises, it was assumed that the functions that were applied over vectors and lists actually returned a meaningful result. For example, the tolower() function simply returns the strings with the characters in lowercase. This won't always be the case. Suppose you want to display the structure of every element of a list. You could use the str() function for this, which returns NULL:

```
lapply(list(1, "a", TRUE), str)
```

```
##  num 1
##  chr "a"
##  logi TRUE

## [[1]]
## NULL
##
## [[2]]
## NULL
##
## [[3]]
## NULL
```

This call actually returns a list, the same size as the input list, containing all NULL values. On the other hand calling

```
str(TRUE)
```

```
##  logi TRUE
```

on its own prints only the structure of the logical to the console, not NULL. That's because str() uses invisible() behind the scenes, which returns an invisible copy of the return value, NULL in this case. This prevents it from being printed when the result of str() is not assigned.

What will the following code chunk return (split_low is already available in the workspace)? Try to reason about the result before simply executing it in the console!

```
lapply(split_low, function(x) {
  if (nchar(x[1]) > 5) {
    return(NULL)
  } else {
    return(x[2])
  }
})
```

```
## [[1]]
## [1] "1777"
##
```

```
## [[2]]
## [1] "1702"
##
## [[3]]
## NULL
##
## [[4]]
## NULL
```