

10. lapply and sapply (xtra)

pkm_edu

07-06-2022

```
options(knitr.duplicate.label = "allow")
```

10. lapply and sapply

Author: swirl Team

Acknowledgements: R Language Concepts and code questions are used here from the swirl package. <https://www.r-project.org/nosvn/pandoc/swirl.html>

In this lesson, you'll learn how to use lapply() and sapply(), the two most important members of R's *apply family of functions, also known as loop functions.

These powerful functions, along with their close relatives (vapply() and tapply(), among others) offer a concise and convenient means of implementing the Split-Apply-Combine strategy for data analysis.

Each of the *apply functions will SPLIT up some data into smaller pieces, APPLY a function to each piece, then COMBINE the results. A more detailed discussion of this strategy is found in Hadley Wickham's Journal of Statistical Software paper titled 'The Split-Apply-Combine Strategy for Data Analysis'.

Throughout this lesson, we'll use the Flags dataset from the UCI Machine Learning Repository. This dataset contains details of various nations and their flags. More information may be found here: <http://archive.ics.uci.edu/ml/datasets/Flags>

Let's jump right in so you can get a feel for how these special functions work!

I've stored the dataset in a variable called flags. Type head(flags) to preview the first six lines (i.e. the 'head') of the

```
# {r setup, include=FALSE, echo=FALSE}
```

```
require("knitr")
```

```
## Loading required package: knitr
```

```
opts_knit$set(root.dir = "C:/r-basics/Data")
```

```
getwd()
```

```
## [1] "C:/r-basics/Data"
```

```
setwd("c:/r-basics/Data")
load('flags.Rdata')
flags <- flags
```

```
library(tidyverse)
```

```
## -- Attaching packages ----- tidyverse 1.3.1 --
```

```
## v ggplot2 3.3.6      v purrr  0.3.4
## v tibble  3.1.7      v dplyr  1.0.9
## v tidyr   1.2.0      v stringr 1.4.0
## v readr   2.1.2      v forcats 0.5.1
```

```
## -- Conflicts ----- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()     masks stats::lag()
```

```
flags %>% dplyr::select(name, landmass, zone) %>%
  dplyr::filter(row_number() %in% c(1,2,3,4,5,6,7))
```

```
##           name landmass zone
## 1  Afghanistan      5     1
## 2    Albania       3     1
## 3    Algeria       4     1
## 4 American-Samoa     6     3
## 5    Andorra       3     1
## 6    Angola       4     2
## 7  Anguilla       1     4
```

```
dim(flags)
```

```
## [1] 194  30
```

This tells us that there are 194 rows, or observations, and 30 columns, or variables. Each observation is a country and each variable describes some characteristic of that country or its flag. To open a more complete description of the dataset in a separate text file, type `viewinfo()` when you are back at the prompt .

As with any dataset, we'd like to know in what format the variables have been stored. In other words, what is the 'class' of each variable? What happens if we do `class(flags)`? Try it out.

```
class(flags)
```

```
## [1] "data.frame"
```

That just tells us that the entire dataset is stored as a 'data.frame', which doesn't answer our question. What we really need is to call the `class()` function on each individual column. While we could do this manually (i.e. one column at a time) it's much faster if we can automate the process. Sounds like a loop!

The `lapply()` function takes a list as input, applies a function to each element of the list, then returns a list of the same length as the original one. Since a data frame is really just a list of vectors (you can see this

with `as.list(flags)`), we can use `lapply()` to apply the `class()` function to each column of the flags dataset. Let's see it in action!

Type `cls_list <- lapply(flags, class)` to apply the `class()` function to each column of the flags dataset and store the result in a variable called `cls_list`. Note that you just supply the name of the function you want to apply (i.e. `class`), without the usual parentheses after it.

```
cls_list <- lapply(flags, class)
```

Type `cls_list` to view the result.

```
cls_list
```

```
## $name
## [1] "character"
##
## $landmass
## [1] "integer"
##
## $zone
## [1] "integer"
##
## $area
## [1] "integer"
##
## $population
## [1] "integer"
##
## $language
## [1] "integer"
##
## $religion
## [1] "integer"
##
## $bars
## [1] "integer"
##
## $stripes
## [1] "integer"
##
## $colours
## [1] "integer"
##
## $red
## [1] "integer"
##
## $green
## [1] "integer"
##
## $blue
## [1] "integer"
##
## $gold
## [1] "integer"
```

```
##
## $white
## [1] "integer"
##
## $black
## [1] "integer"
##
## $orange
## [1] "integer"
##
## $mainhue
## [1] "character"
##
## $circles
## [1] "integer"
##
## $crosses
## [1] "integer"
##
## $saltires
## [1] "integer"
##
## $quarters
## [1] "integer"
##
## $sunstars
## [1] "integer"
##
## $crescent
## [1] "integer"
##
## $triangle
## [1] "integer"
##
## $icon
## [1] "integer"
##
## $animate
## [1] "integer"
##
## $text
## [1] "integer"
##
## $topleft
## [1] "character"
##
## $botright
## [1] "character"
```

The 'l' in 'lapply' stands for 'list'. Type `class(cls_list)` to confirm that `lapply()` returned a list.

```
class(cls_list)
```

```
## [1] "list"
```

As expected, we got a list of length 30 – one element for each variable/column. The output would be considerably more compact if we could represent it as a vector instead of a list.

You may remember from a previous lesson that lists are most helpful for storing multiple classes of data. In this case, since every element of the list returned by `lapply()` is a character vector of length one (i.e. “integer” and “vector”), `cls_list` can be simplified to a character vector. To do this manually, type `as.character(cls_list)`.

```
as.character(cls_list)
```

```
## [1] "character" "integer"    "integer"    "integer"    "integer"    "integer"
## [7] "integer"    "integer"    "integer"    "integer"    "integer"    "integer"
## [13] "integer"    "integer"    "integer"    "integer"    "integer"    "character"
## [19] "integer"    "integer"    "integer"    "integer"    "integer"    "integer"
## [25] "integer"    "integer"    "integer"    "integer"    "character"  "character"
```

`sapply()` allows you to automate this process by calling `lapply()` behind the scenes, but then attempting to simplify (hence the ‘s’ in ‘sapply’) the result for you. Use `sapply()` the same way you used `lapply()` to get the class of each column of the flags dataset and store the result in `cls_vect`. If you need help, type `?sapply` to bring up the documentation.

```
?sapply
```

```
## starting httpd help server ... done
```

```
cls_vect <- sapply(flags, class)
```

```
class(cls_vect)
```

```
## [1] "character"
```

In general, if the result is a list where every element is of length one, then `sapply()` returns a vector. If the result is a list where every element is a vector of the same length (1), `sapply()` returns a matrix. If `sapply()` can’t figure things out, then it just returns a list, no different from what `lapply()` would give you.

Let’s practice using `lapply()` and `sapply()` some more!

Columns 11 through 17 of our dataset are indicator variables, each representing a different color. The value of the indicator variable is 1 if the color is present in a country’s flag and 0 otherwise.

Therefore, if we want to know the total number of countries (in our dataset) with, for example, the color orange on their flag, we can just add up all of the 1s and 0s in the ‘orange’ column. Try `sum(flags$orange)` to see this.

```
sum(flags$orange)
```

```
## [1] 26
```

Now we want to repeat this operation for each of the colors recorded in the dataset.

First, use `flag_colors <- flags[, 11:17]` to extract the columns containing the color data and store them in a new data frame called `flag_colors`. (Note the comma before 11:17. This subsetting command tells R that we want all rows, but only columns 11 through 17.)

```
flag_colors <- flags[, 11:17]
```

```
head(flag_colors)
```

```
##   red green blue gold white black orange
## 1   1    1    0    1    1    1    0
## 2   1    0    0    1    0    1    0
## 3   1    1    0    0    1    0    0
## 4   1    0    1    1    1    0    1
## 5   1    0    1    1    0    0    0
## 6   1    0    0    1    0    1    0
```

To get a list containing the sum of each column of `flag_colors`, call the `lapply()` function with two arguments. The first argument is the object over which we are looping (i.e. `flag_colors`) and the second argument is the name of the function we wish to apply to each column (i.e. `sum`). Remember that the second argument is just the name of the function with no parentheses, etc.

```
lapply(flag_colors, sum)
```

```
## $red
## [1] 153
##
## $green
## [1] 91
##
## $blue
## [1] 99
##
## $gold
## [1] 91
##
## $white
## [1] 146
##
## $black
## [1] 52
##
## $orange
## [1] 26
```

This tells us that of the 194 flags in our dataset, 153 contain the color red, 91 contain green, 99 contain blue, and so on. The result is a list, since `lapply()` always returns a list. Each element of this list is of length one, so the result can be simplified to a vector by calling `sapply()` instead of `lapply()`. Try it now.

```
sapply(flag_colors, sum)
```

```
##   red green blue gold white black orange
##  153    91   99   91  146    52    26
```

Perhaps it's more informative to find the proportion of flags (out of 194) containing each color. Since each column is just a bunch of 1s and 0s, the arithmetic mean of each column will give us the proportion of 1s.

(If it's not clear why, think of a simpler situation where you have three 1s and two 0s – $(1 + 1 + 1 + 0 + 0)/5 = 3/5 = 0.6$).

Use `sapply()` to apply the `mean()` function to each column of `flag_colors`. Remember that the second argument to `sapply()` should just specify the name of the function (i.e. `mean`) that you want to apply.

```
sapply(flag_colors, mean)
```

```
##      red      green      blue      gold      white      black      orange
## 0.7886598 0.4690722 0.5103093 0.4690722 0.7525773 0.2680412 0.1340206
```

In the examples we've looked at so far, `sapply()` has been able to simplify the result to vector. That's because each element of the list returned by `lapply()` was a vector of length one. Recall that `sapply()` instead returns a matrix when each element of the list returned by `lapply()` is a vector of the same length (1).

To illustrate this, let's extract columns 19 through 23 from the `flags` dataset and store the result in a new data frame called `flag_shapes`. `flag_shapes <- flags[, 19:23]` will do it.

`flag_shapes <- flags[, 19:23]` will save columns 19 through 23 in a new variable.

```
flag_shapes <- flags[, 19:23]
```

Each of these columns (i.e. variables) represents the number of times a particular shape or design appears on a country's flag. We are interested in the minimum and maximum number of times each shape or design appears.

The `range()` function returns the minimum and maximum of its first argument, which should be a numeric vector. Use `lapply()` to apply the `range` function to each column of `flag_shapes`. Don't worry about storing the result in a new variable. By now, we know that `lapply()` always returns a list.

```
lapply(flag_shapes, range)
```

```
## $circles
## [1] 0 4
##
## $crosses
## [1] 0 2
##
## $saltires
## [1] 0 1
##
## $quarters
## [1] 0 4
##
## $sunstars
## [1] 0 50
```

Do the same operation, but using `sapply()` and store the result in a variable called `shape_mat`.

`shape_mat <- sapply(flag_shapes, range)` will apply the `range()` function to each column of `flag_shapes` and store the result in `shape_mat`.

```
shape_mat <- sapply(flag_shapes, range)
```

Each column of `shape_mat` gives the minimum (row 1) and maximum (row 2) number of times its respective shape appears in different flags.

Use the `class()` function to confirm that `shape_mat` is a matrix.

```
is.matrix(shape_mat)
```

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

As we've seen, `sapply()` always attempts to simplify the result given by `lapply()`. It has been successful in doing so for each of the examples we've looked at so far. Let's look at an example where `sapply()` can't figure out how to simplify the result and thus returns a list, no different from `lapply()`.

When given a vector, the `unique()` function returns a vector with all duplicate elements removed. In other words, `unique()` returns a vector of only the 'unique' elements. To see how it works, try `unique(c(3, 4, 5, 5, 5, 6, 6))`.

```
unique(c(3, 4, 5, 5, 5, 6, 6))
```

```
## [1] 3 4 5 6
```

We want to know the unique values for each variable in the flags dataset. To accomplish this, use `lapply()` to apply the `unique()` function to each column in the flags dataset, storing the result in a variable called `unique_vals`.

```
unique_vals <- lapply(flags, unique)
```

Print the value of `unique_vals` to the console.

```
unique_vals
```

```
## $name
## [1] "Afghanistan"      "Albania"
## [3] "Algeria"          "American-Samoa"
## [5] "Andorra"          "Angola"
## [7] "Anguilla"         "Antigua-Barbuda"
## [9] "Argentina"        "Argentine"
## [11] "Australia"        "Austria"
## [13] "Bahamas"          "Bahrain"
## [15] "Bangladesh"       "Barbados"
## [17] "Belgium"          "Belize"
## [19] "Benin"            "Bermuda"
## [21] "Bhutan"           "Bolivia"
## [23] "Botswana"         "Brazil"
## [25] "British-Virgin-Isles" "Brunei"
## [27] "Bulgaria"         "Burkina"
## [29] "Burma"            "Burundi"
## [31] "Cameroon"         "Canada"
## [33] "Cape-Verde-Islands" "Cayman-Islands"
```


## [35]	"Central-African-Republic"	"Chad"
## [37]	"Chile"	"China"
## [39]	"Colombia"	"Comorro-Islands"
## [41]	"Congo"	"Cook-Islands"
## [43]	"Costa-Rica"	"Cuba"
## [45]	"Cyprus"	"Czechoslovakia"
## [47]	"Denmark"	"Djibouti"
## [49]	"Dominica"	"Dominican-Republic"
## [51]	"Ecuador"	"Egypt"
## [53]	"El-Salvador"	"Equatorial-Guinea"
## [55]	"Ethiopia"	"Faeroes"
## [57]	"Falklands-Malvinas"	"Fiji"
## [59]	"Finland"	"France"
## [61]	"French-Guiana"	"French-Polynesia"
## [63]	"Gabon"	"Gambia"
## [65]	"Germany-DDR"	"Germany-FRG"
## [67]	"Ghana"	"Gibraltar"
## [69]	"Greece"	"Greenland"
## [71]	"Grenada"	"Guam"
## [73]	"Guatemala"	"Guinea"
## [75]	"Guinea-Bissau"	"Guyana"
## [77]	"Haiti"	"Honduras"
## [79]	"Hong-Kong"	"Hungary"
## [81]	"Iceland"	"India"
## [83]	"Indonesia"	"Iran"
## [85]	"Iraq"	"Ireland"
## [87]	"Israel"	"Italy"
## [89]	"Ivory-Coast"	"Jamaica"
## [91]	"Japan"	"Jordan"
## [93]	"Kampuchea"	"Kenya"
## [95]	"Kiribati"	"Kuwait"
## [97]	"Laos"	"Lebanon"
## [99]	"Lesotho"	"Liberia"
## [101]	"Libya"	"Liechtenstein"
## [103]	"Luxembourg"	"Malagasy"
## [105]	"Malawi"	"Malaysia"
## [107]	"Maldives-Islands"	"Mali"
## [109]	"Malta"	"Marianas"
## [111]	"Mauritania"	"Mauritius"
## [113]	"Mexico"	"Micronesia"
## [115]	"Monaco"	"Mongolia"
## [117]	"Montserrat"	"Morocco"
## [119]	"Mozambique"	"Nauru"
## [121]	"Nepal"	"Netherlands"
## [123]	"Netherlands-Antilles"	"New-Zealand"
## [125]	"Nicaragua"	"Niger"
## [127]	"Nigeria"	"Niue"
## [129]	"North-Korea"	"North-Yemen"
## [131]	"Norway"	"Oman"
## [133]	"Pakistan"	"Panama"
## [135]	"Papua-New-Guinea"	"Paraguay"
## [137]	"Peru"	"Philippines"
## [139]	"Poland"	"Portugal"
## [141]	"Puerto-Rico"	"Qatar"

```

## [143] "Romania"
## [145] "San-Marino"
## [147] "Saudi-Arabia"
## [149] "Seychelles"
## [151] "Singapore"
## [153] "Somalia"
## [155] "South-Korea"
## [157] "Spain"
## [159] "St-Helena"
## [161] "St-Lucia"
## [163] "Sudan"
## [165] "Swaziland"
## [167] "Switzerland"
## [169] "Taiwan"
## [171] "Thailand"
## [173] "Tonga"
## [175] "Tunisia"
## [177] "Turks-Cocos-Islands"
## [179] "UAE"
## [181] "UK"
## [183] "US-Virgin-Isles"
## [185] "USSR"
## [187] "Vatican-City"
## [189] "Vietnam"
## [191] "Yugoslavia"
## [193] "Zambia"
##
## "Rwanda"
## "Sao-Tome"
## "Senegal"
## "Sierra-Leone"
## "Soloman-Islands"
## "South-Africa"
## "South-Yemen"
## "Sri-Lanka"
## "St-Kitts-Nevis"
## "St-Vincent"
## "Surinam"
## "Sweden"
## "Syria"
## "Tanzania"
## "Togo"
## "Trinidad-Tobago"
## "Turkey"
## "Tuvalu"
## "Uganda"
## "Uruguay"
## "USA"
## "Vanuatu"
## "Venezuela"
## "Western-Samoa"
## "Zaire"
## "Zimbabwe"
##
## $landmass
## [1] 5 3 4 6 1 2
##
## $zone
## [1] 1 3 2 4
##
## $area
## [1] 648 29 2388 0 1247 2777 7690 84 19 1 143 31
## [13] 23 113 47 1099 600 8512 6 111 274 678 28 474
## [25] 9976 4 623 1284 757 9561 1139 2 342 51 115 9
## [37] 128 43 22 49 284 1001 21 1222 12 18 337 547
## [49] 91 268 10 108 249 239 132 2176 109 246 36 215
## [61] 112 93 103 3268 1904 1648 435 70 301 323 11 372
## [73] 98 181 583 236 30 1760 3 587 118 333 1240 1031
## [85] 1973 1566 447 783 140 41 1267 925 121 195 324 212
## [97] 804 76 463 407 1285 300 313 92 237 26 2150 196
## [109] 72 637 1221 99 288 505 66 2506 63 17 450 185
## [121] 945 514 57 5 164 781 245 178 9363 22402 15 912
## [133] 256 905 753 391
##
## $population
## [1] 16 3 20 0 7 28 15 8 90 10 1 6 119 9 35
## [16] 4 24 2 11 1008 5 47 31 54 17 61 14 684 157 39
## [31] 57 118 13 77 12 56 18 84 48 36 22 29 38 49 45
## [46] 231 274 60
##
## $language

```

```

## [1] 10 6 8 1 2 4 3 5 7 9
##
## $religion
## [1] 2 6 1 0 5 3 4 7
##
## $bars
## [1] 0 2 3 1 5
##
## $stripes
## [1] 3 0 2 1 5 9 11 14 4 6 13 7
##
## $colours
## [1] 5 3 2 8 6 4 7 1
##
## $red
## [1] 1 0
##
## $green
## [1] 1 0
##
## $blue
## [1] 0 1
##
## $gold
## [1] 1 0
##
## $white
## [1] 1 0
##
## $black
## [1] 1 0
##
## $orange
## [1] 0 1
##
## $mainhue
## [1] "green" "red" "blue" "gold" "white" "orange" "black" "brown"
##
## $circles
## [1] 0 1 4 2
##
## $crosses
## [1] 0 1 2
##
## $saltires
## [1] 0 1
##
## $quarters
## [1] 0 1 4
##
## $sunstars
## [1] 1 0 6 22 14 3 4 5 15 10 7 2 9 50
##
## $crescent

```

```
## [1] 0 1
##
## $triangle
## [1] 0 1
##
## $icon
## [1] 1 0
##
## $animate
## [1] 0 1
##
## $text
## [1] 0 1
##
## $topleft
## [1] "black" "red" "green" "blue" "white" "orange" "gold"
##
## $botright
## [1] "green" "red" "white" "black" "blue" "gold" "orange" "brown"
```

Since `unique_vals` is a list, you can use what you've learned to determine the length of each element of `unique_vals` (i.e. the number of unique values for each variable). Simplify the result, if possible. Hint: Apply the `length()` function to each element of `unique_vals`.

```
sapply(unique_vals, length)
```

```
##      name  landmass      zone      area population  language  religion
##      194       6       4      136       48       10         8
##      bars  stripes  colours      red      green      blue      gold
##       5      12       8       2       2       2       2
##      white  black    orange  mainhue  circles  crosses  saltires
##       2       2       2       8       4       3       2
##  quarters  sunstars  crescent  triangle      icon  animate      text
##       3      14       2       2       2       2       2
##  topleft  botright
##       7       8
```

The fact that the elements of the `unique_vals` list are all vectors of *different* length poses a problem for `sapply()`, since there's no obvious way of simplifying the result.

Use `sapply()` to apply the `unique()` function to each column of the `flags` dataset to see that you get the same unsimplified list that you got from `lapply()`.

```
sapply(flags, unique)
```

```
## $name
## [1] "Afghanistan" "Albania"
## [3] "Algeria" "American-Samoa"
## [5] "Andorra" "Angola"
## [7] "Anguilla" "Antigua-Barbuda"
## [9] "Argentina" "Argentine"
## [11] "Australia" "Austria"
## [13] "Bahamas" "Bahrain"
```

## [15]	"Bangladesh"	"Barbados"
## [17]	"Belgium"	"Belize"
## [19]	"Benin"	"Bermuda"
## [21]	"Bhutan"	"Bolivia"
## [23]	"Botswana"	"Brazil"
## [25]	"British-Virgin-Isles"	"Brunei"
## [27]	"Bulgaria"	"Burkina"
## [29]	"Burma"	"Burundi"
## [31]	"Cameroon"	"Canada"
## [33]	"Cape-Verde-Islands"	"Cayman-Islands"
## [35]	"Central-African-Republic"	"Chad"
## [37]	"Chile"	"China"
## [39]	"Colombia"	"Comorro-Islands"
## [41]	"Congo"	"Cook-Islands"
## [43]	"Costa-Rica"	"Cuba"
## [45]	"Cyprus"	"Czechoslovakia"
## [47]	"Denmark"	"Djibouti"
## [49]	"Dominica"	"Dominican-Republic"
## [51]	"Ecuador"	"Egypt"
## [53]	"El-Salvador"	"Equatorial-Guinea"
## [55]	"Ethiopia"	"Faeroes"
## [57]	"Falklands-Malvinas"	"Fiji"
## [59]	"Finland"	"France"
## [61]	"French-Guiana"	"French-Polynesia"
## [63]	"Gabon"	"Gambia"
## [65]	"Germany-DDR"	"Germany-FRG"
## [67]	"Ghana"	"Gibraltar"
## [69]	"Greece"	"Greenland"
## [71]	"Grenada"	"Guam"
## [73]	"Guatemala"	"Guinea"
## [75]	"Guinea-Bissau"	"Guyana"
## [77]	"Haiti"	"Honduras"
## [79]	"Hong-Kong"	"Hungary"
## [81]	"Iceland"	"India"
## [83]	"Indonesia"	"Iran"
## [85]	"Iraq"	"Ireland"
## [87]	"Israel"	"Italy"
## [89]	"Ivory-Coast"	"Jamaica"
## [91]	"Japan"	"Jordan"
## [93]	"Kampuchea"	"Kenya"
## [95]	"Kiribati"	"Kuwait"
## [97]	"Laos"	"Lebanon"
## [99]	"Lesotho"	"Liberia"
## [101]	"Libya"	"Liechtenstein"
## [103]	"Luxembourg"	"Malagasy"
## [105]	"Malawi"	"Malaysia"
## [107]	"Maldives-Islands"	"Mali"
## [109]	"Malta"	"Marianas"
## [111]	"Mauritania"	"Mauritius"
## [113]	"Mexico"	"Micronesia"
## [115]	"Monaco"	"Mongolia"
## [117]	"Montserrat"	"Morocco"
## [119]	"Mozambique"	"Nauru"
## [121]	"Nepal"	"Netherlands"

```

## [123] "Netherlands-Antilles"    "New-Zealand"
## [125] "Nicaragua"               "Niger"
## [127] "Nigeria"                 "Niue"
## [129] "North-Korea"             "North-Yemen"
## [131] "Norway"                   "Oman"
## [133] "Pakistan"                 "Panama"
## [135] "Papua-New-Guinea"        "Parguay"
## [137] "Peru"                     "Philippines"
## [139] "Poland"                   "Portugal"
## [141] "Puerto-Rico"            "Qatar"
## [143] "Romania"                  "Rwanda"
## [145] "San-Marino"               "Sao-Tome"
## [147] "Saudi-Arabia"             "Senegal"
## [149] "Seychelles"              "Sierra-Leone"
## [151] "Singapore"               "Soloman-Islands"
## [153] "Somalia"                  "South-Africa"
## [155] "South-Korea"             "South-Yemen"
## [157] "Spain"                    "Sri-Lanka"
## [159] "St-Helena"               "St-Kitts-Nevis"
## [161] "St-Lucia"                "St-Vincent"
## [163] "Sudan"                    "Surinam"
## [165] "Swaziland"                "Sweden"
## [167] "Switzerland"             "Syria"
## [169] "Taiwan"                   "Tanzania"
## [171] "Thailand"                 "Togo"
## [173] "Tonga"                    "Trinidad-Tobago"
## [175] "Tunisia"                  "Turkey"
## [177] "Turks-Cocos-Islands"     "Tuvalu"
## [179] "UAE"                      "Uganda"
## [181] "UK"                       "Uruguay"
## [183] "US-Virgin-Isles"         "USA"
## [185] "USSR"                     "Vanuatu"
## [187] "Vatican-City"            "Venezuela"
## [189] "Vietnam"                  "Western-Samoa"
## [191] "Yugoslavia"              "Zaire"
## [193] "Zambia"                   "Zimbabwe"
##
## $landmass
## [1] 5 3 4 6 1 2
##
## $zone
## [1] 1 3 2 4
##
## $area
## [1] 648 29 2388 0 1247 2777 7690 84 19 1 143 31
## [13] 23 113 47 1099 600 8512 6 111 274 678 28 474
## [25] 9976 4 623 1284 757 9561 1139 2 342 51 115 9
## [37] 128 43 22 49 284 1001 21 1222 12 18 337 547
## [49] 91 268 10 108 249 239 132 2176 109 246 36 215
## [61] 112 93 103 3268 1904 1648 435 70 301 323 11 372
## [73] 98 181 583 236 30 1760 3 587 118 333 1240 1031
## [85] 1973 1566 447 783 140 41 1267 925 121 195 324 212
## [97] 804 76 463 407 1285 300 313 92 237 26 2150 196
## [109] 72 637 1221 99 288 505 66 2506 63 17 450 185

```

```

## [121] 945 514 57 5 164 781 245 178 9363 22402 15 912
## [133] 256 905 753 391
##
## $population
## [1] 16 3 20 0 7 28 15 8 90 10 1 6 119 9 35
## [16] 4 24 2 11 1008 5 47 31 54 17 61 14 684 157 39
## [31] 57 118 13 77 12 56 18 84 48 36 22 29 38 49 45
## [46] 231 274 60
##
## $language
## [1] 10 6 8 1 2 4 3 5 7 9
##
## $religion
## [1] 2 6 1 0 5 3 4 7
##
## $bars
## [1] 0 2 3 1 5
##
## $stripes
## [1] 3 0 2 1 5 9 11 14 4 6 13 7
##
## $colours
## [1] 5 3 2 8 6 4 7 1
##
## $red
## [1] 1 0
##
## $green
## [1] 1 0
##
## $blue
## [1] 0 1
##
## $gold
## [1] 1 0
##
## $white
## [1] 1 0
##
## $black
## [1] 1 0
##
## $orange
## [1] 0 1
##
## $mainhue
## [1] "green" "red" "blue" "gold" "white" "orange" "black" "brown"
##
## $circles
## [1] 0 1 4 2
##
## $crosses
## [1] 0 1 2
##

```

```
## $saltires
## [1] 0 1
##
## $quarters
## [1] 0 1 4
##
## $sunstars
## [1] 1 0 6 22 14 3 4 5 15 10 7 2 9 50
##
## $crescent
## [1] 0 1
##
## $triangle
## [1] 0 1
##
## $icon
## [1] 1 0
##
## $animate
## [1] 0 1
##
## $text
## [1] 0 1
##
## $topleft
## [1] "black" "red" "green" "blue" "white" "orange" "gold"
##
## $botright
## [1] "green" "red" "white" "black" "blue" "gold" "orange" "brown"
```

Occasionally, you may need to apply a function that is not yet defined, thus requiring you to write your own. Writing functions in R is beyond the scope of this lesson, but let's look at a quick example of how you might do so in the context of loop functions.

Pretend you are interested in only the second item from each element of the `unique_vals` list that you just created. Since each element of the `unique_vals` list is a vector and we're not aware of any built-in function in R that returns the second element of a vector, we will construct our own function.

`lapply(unique_vals, function(elem) elem[2])` will return a list containing the second item from each element of the `unique_vals` list. Note that our function takes one argument, `elem`, which is just a 'dummy variable' that takes on the value of each element of `unique_vals`, in turn.

```
lapply(unique_vals, function(elem) elem[2])
```

```
## $name
## [1] "Albania"
##
## $landmass
## [1] 3
##
## $zone
## [1] 3
##
## $area
```



```
## [1] 29
##
## $population
## [1] 3
##
## $language
## [1] 6
##
## $religion
## [1] 6
##
## $bars
## [1] 2
##
## $stripes
## [1] 0
##
## $colours
## [1] 3
##
## $red
## [1] 0
##
## $green
## [1] 0
##
## $blue
## [1] 1
##
## $gold
## [1] 0
##
## $white
## [1] 0
##
## $black
## [1] 0
##
## $orange
## [1] 1
##
## $mainhue
## [1] "red"
##
## $circles
## [1] 1
##
## $crosses
## [1] 1
##
## $saltires
## [1] 1
##
## $quarters
```

```
## [1] 1
##
## $sunstars
## [1] 0
##
## $crescent
## [1] 1
##
## $triangle
## [1] 1
##
## $icon
## [1] 0
##
## $animate
## [1] 1
##
## $text
## [1] 1
##
## $topleft
## [1] "red"
##
## $botright
## [1] "red"
```

... Our function has no name and disappears as soon as `lapply()` is done using it. So-called ‘anonymous functions’ can be very useful when one of R’s built-in functions isn’t an option.

In this lesson, you learned how to use the powerful `lapply()` and `sapply()` functions to apply an operation over the elements of a list. In the next lesson, we’ll take a look at some close relatives of `lapply()` and `sapply()`.