



INTERMEDIATE R

# **lapply**

# NYC: for

```
> nyc <- list(pop = 8405837,  
+                 boroughs = c("Manhattan", "Bronx", "Brooklyn",  
+                                         "Queens", "Staten Island"),  
+                 capital = FALSE)  
  
> for(info in nyc) {  
+   print(class(info))  
+ }  
[1] "numeric"  
[1] "character"  
[1] "logical"
```

# NYC: 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"
```

Under the hood, lapply iterates over the list, nyc, and on each element of the list, applied the function class().

The output of lapply includes the results of calling the class function over each of the list elements.

# Cities: for

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

> num_chars <- c()  create an empty vector
> for(i in 1:length(cities)) {
  num_chars[i] <- nchar(cities[i])
}

> num_chars
[1]  8  5  6  5 14  9
```

# Cities: lapply()

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

```
> lapply(cities, nchar)  
[[1]]  
[1] 8
```

```
[[2]]  
[1] 5
```

...

```
[[6]]  
[1] 9
```

The output is a list, although the input was a vector.  
lapply function always returns a list, irrespective of the input data structure.

# Cities: lapply()

```
> cities <- c("New York", "Paris", "London", "Tokyo",  
  "Rio de Janeiro", "Cape Town")  
  
> unlist(lapply(cities, nchar)) [1] 8 5 6 5 14 9
```

If you want to convert this list to a vector, you can simply wrap this `lapply` function inside the `unlist()` functions which turns a list into a vector.

# Oil

```
> oil_prices <- list(2.37, 2.49, 2.18, 2.22, 2.47, 2.32)
> triple <- function(x) {
  3 * x
}
> result <- lapply(oil_prices, triple)
> str(result)
List of 6
$ : num 7.11
$ : num 7.47
$ : num 6.54
$ : num 6.66
$ : num 7.41
$ : num 6.96
> unlist(result)
[1] 7.11 7.47 6.54 6.66 7.41 6.96
```

# Oil

`lapply` allows you to add additional arguments to the function: just include them right after the function you want to apply to your list or vector.

```
> 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

> times4 <- lapply(oil_prices, multiply, factor = 4)
> unlist(times4)
[1] 9.48 9.96 8.72 8.88 9.88 9.28
```



INTERMEDIATE R

# Let's practice!



INTERMEDIATE R

# sapply

# lapply()

- Apply function over list or vector
- Function can return R objects of different classes
- List necessary to store heterogeneous content
- However, often homogeneous content

# Cities: lapply()

```
> cities <- c("New York", "Paris", "London", "Tokyo",
  "Rio de Janeiro", "Cape Town")  
  
> result <- lapply(cities, nchar)  
  
> str(result)  
List of 6  
 $ : int 8  
 $ : int 5  
 $ : int 6  
 $ : int 5  
 $ : int 14  
 $ : int 9  
  
> unlist(lapply(cities, nchar))  
[1] 8 5 6 5 14 9
```

As a result, we obtained a list with the length of each of the cities names. But these values could well fit into a simple vector as well.

# Cities: sapply()

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

```
> unlist(lapply(cities, nchar))  
[1] 8 5 6 5 14 9
```

```
> sapply(cities, nchar) There is an easier way to tackle the case in which all the results have the same type.
```

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

sapply is short for “simplify apply”

```
> sapply(cities, nchar, USE.NAMES = FALSE)  
[1] 8 5 6 5 14 9
```

**USE.NAMES is TRUE by default**

# Cities: sapply()

```
> 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"  
  
> 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"
```

# Unable to simplify?

There are cases in which the function you want to apply does not always return a vector of the same length at all times.

```
> unique_letters <- function(name) {  
  name <- gsub(" ", "", name)  
  letters <- strsplit(name, split = "")[[1]]  
  unique(letters)  
}  
  
> unique_letters("London")  
[1] "L" "o" "n" "d"
```

# Unable to simplify?

```
> lapply(cities, unique_letters)
[[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"
...
...
```

```
> sapply(cities, unique_letters)
$`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"
...
...
```

**sapply did not simplify**  
**Can be dangerous!**



INTERMEDIATE R

**Let's practice!**



INTERMEDIATE R

# vapply

# Recap

- **lapply()**  
apply function over list or vector  
output = list
- **sapply()** This can be dangerous, because the behavior of sapply's output depends on the specifics of the data we are using.  
apply function over list or vector  
try to simplify list to array
- **vapply()** When using vapply, you have to explicitly say what the type of the return value will be.  
apply function over list or vector  
explicitly specify output format

# sapply() & vapply()

```
> cities <- c("New York", "Paris", "London", "Tokyo",
  "Rio de Janeiro", "Cape Town")  
  
> sapply(cities, nchar)  
New York    Paris    London    Tokyo    Rio de Janeiro    Cape Town  
          8         5         6         5           14            9
```

vapply(x, FUN, **FUN.VALUE**, ..., USE.NAMES = TRUE)



**FUN.VALUE** is new here. This argument should be a general template for the return value of **FUN**, the function that you want to apply over the input **X**.

```
> vapply(cities, nchar, numeric(1))  
New York    Paris    London    Tokyo    Rio de Janeiro    Cape Town  
          8         5         6         5           14            9
```

nchar is a function that returns a single number, which is a numeric vector of length 1  
numeric(1) tells the vapply function that nchar() should return a single numerical value

This pre-specification of **FUN**'s return value makes vapply a safer alternative to sapply.

# vapply()

```
> first_and_last <- function(name) {  
  name <- gsub(" ", "", name)  
  letters <- strsplit(name, split = "")[[1]]  
  return(c(first = min(letters), last = max(letters)))  
}  
  
> 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"  
  
> vapply(cities, first_and_last, character(2))  
      New York Paris London Tokyo Rio de Janeiro Cape Town  
first "e"      "a"    "d"    "k"    "a"          "a"  
last  "Y"      "s"    "o"    "y"    "R"          "w"
```

# vapply() errors

```
> vapply(cities, first_and_last, character(2))
  New York Paris London Tokyo Rio de Janeiro Cape Town
first "e"      "a"    "d"    "k"    "a"                  "a"
last  "Y"      "s"    "o"    "y"    "R"                  "w"
```

```
> vapply(cities, first_and_last, character(1)) R complains
Error in vapply(cities, first_and_last, character(1)) :
  values must be length 1,
  but FUN(X[[1]]) result is length 2
```

```
> vapply(cities, first_and_last, numeric(2))
Error in vapply(cities, first_and_last, numeric(2)) :
  values must be type 'double',
  but FUN(X[[1]]) result is type 'character'
```

# unique\_letters()

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

# vapply() > sapply()

```
> sapply(cities, unique_letters)
$`New York`
[1] "N" "e" "w" "Y" "o" "r" "k"
```

...

```
$`Cape Town`
[1] "C" "a" "p" "e" "T" "o" "w" "n"
```

**vapply() is safer than sapply()!**

```
> vapply(cities, unique_letters, character(4))
Error in vapply(cities, unique_letters, character(4)) :
  values must be length 4,
  but FUN(X[[1]]) result is length 7
```

logical(1)



INTERMEDIATE R

**Let's practice!**