SLOVENSKÁ TECHNICKÁ
UNIVERZITA V BRATISLAVE

# Loop Functions

Dr.Sc. Oleksii Yehorchenkov
Department of Spatial Planning

Funded by
the European Union
NextGenerationEU

# Looping on the Command Line

Writing for, while loops is useful when programming but not particularly easy when working interactively on the command line. There are some functions which implement looping to make life easier.

- `lapply`: Loop over a list and evaluate a function on each element
- `sapply`: Same as `lapply` but try to simplify the result
- `apply`: Apply a function over the margins of an array
- `tapply`: Apply a function over subsets of a vector
- `mapply`: Multivariate version of `lapply`

An auxiliary function `split` is also useful, particularly in conjunction with `lapply`.

# lapply

`lapply` takes three arguments: (1) a list `X`; (2) a function (or the name of a function) `FUN`; (3) other arguments via its `...` argument. If `X` is not a list, it will be coerced to a list using `as.list`.

```
1 lapply
```

```
function (X, FUN, ...)
{
    FUN <- match.fun(FUN)
    if (!is.vector(X) || is.object(X))
        X <- as.list(X)
    .Internal(lapply(X, FUN))
}
<bytecode: 0x0000029665ab1de8>
<environment: namespace:base>
```

The actual looping is done internally in C code.

# lapply

`lapply` always returns a list, regardless of the class of the input.

```
1 x <- list(a = 1:5, b = rnorm(10))
2 lapply(x, mean)
```

```
$a
[1] 3

$b
[1] -0.0978843
```

# lapply

```r
x <- 1:4
lapply(x, runif)
```

```
[[1]]
[1] 0.718553

[[2]]
[1] 0.9019804 0.4337659

[[3]]
[1] 0.8769654 0.4291821 0.9463634

[[4]]
[1] 0.05826219 0.23252787 0.48864057 0.36065883
```

# lapply

```r
x <- 1:4
lapply(x, runif, min = 0, max = 10)
```

```
[[1]]
[1] 4.09815

[[2]]
[1] 9.194934 9.531469

[[3]]
[1] 5.381556 6.049687 1.423354

[[4]]
[1] 1.774819 5.284987 6.524983 6.739993
```

# sapply

`sapply` will try to simplify the result of `lapply` if possible.

- If the result is a list where every element is length 1, then a vector is returned
- If the result is a list where every element is a vector of the same length (> 1), a matrix is returned.
- If it can't figure things out, a list is returned

# sapply

```r
x <- list(a = 1:4, b = rnorm(10), c = rnorm(20, 1), d = rnorm(100, 5))
lapply(x, mean)
```

```
$a
[1] 2.5

$b
[1] 0.02027401

$c
[1] 1.17352

$d
[1] 4.8709
```

## The output from `sapply` function:

```r
sapply(x, mean)
```

```
         a          b          c          d
2.50000000 0.02027401 1.17352009 4.87090004
```

# lapply

```r
x <- data.frame(a = 1, b = "a", c = 23, d = TRUE, e = "c")

sapply(x, is.numeric)
```

```
    a     b     c     d     e 
 TRUE FALSE  TRUE FALSE FALSE 
```

# anonymous functions

`lapply` and friends make heavy use of *anonymous* functions.

```
numbers_list <- list(a = 1:5, b = 6:10, c = 11:15)

stats <- sapply(numbers_list, function(x) {
    c(mean = mean(x),
      sum = sum(x),
      std = sd(x))
})

print(stats)
```

```
             a          b          c
mean  3.000000  8.000000 13.000000
sum  15.000000 40.000000 65.000000
std   1.581139  1.581139  1.581139
```

# apply

```
1 str(apply)
```

```
function (X, MARGIN, FUN, ..., simplify = TRUE)
```

- X is an array
- MARGIN is an integer vector indicating which margins should be “retained”.
- FUN is a function to be applied
- … is for other arguments to be passed to FUN

# apply

## apply by columns

```
1 x <- matrix(rnorm(50, 10), 10, 5)
2 apply(x, 2, mean)
```

```
[1] 10.167521 10.296375 10.080430  9.993722 10.274499
```

## apply by rows

```
1 apply(x, 1, sum)
```

```
[1] 47.17666 49.32429 50.06147 53.65689 50.31776 50.29292 53.26562 50.45107
[9] 51.42411 52.15468
```

# col/row sums and means

For sums and means of matrix dimensions, we have some shortcuts.

- `rowSums` = `apply(x, 1, sum)`
- `rowMeans` = `apply(x, 1, mean)`
- `colSums` = `apply(x, 2, sum)`
- `colMeans` = `apply(x, 2, mean)`

# Other Ways to Apply

## Quantiles of the rows of a matrix.

```
1  x <- matrix(rnorm(50, 10), 10, 5)
2  apply(x, 1, quantile, probs = c(0.25, 0.75))
```

```
         [,1]     [,2]     [,3]      [,4]     [,5]      [,6]      [,7]
25%  9.833297 9.152009 9.045668  9.705473 8.971367  9.913912  9.649221
75% 10.324780 9.800468 9.921422 11.078110 9.537355 10.915204 10.224081
         [,8]      [,9]    [,10]
25%  9.796416  9.432373 10.11924
75% 10.372122 10.536586 11.46567
```

# apply

## Average matrix in an array

```r
a <- array(rnorm(2 * 2 * 10), c(2, 2, 10))
apply(a, c(1, 2), mean)
```

```
           [,1]      [,2]
[1,]  0.3725062 0.3878469
[2,] -0.2817413 0.2327591
```

```r
rowMeans(a, dims = 2)
```

```
           [,1]      [,2]
[1,]  0.3725062 0.3878469
[2,] -0.2817413 0.2327591
```

# References

This presentation is based on Coursera course R Programming from John Hopkins University