Хэш-таблицы в R

Хэш-таблицы в R

Строго говоря - их нет)

Хэш-таблицы в R

Строго говоря - их нет)

Списки с заданным names - не хэш-таблицы.

Поиск в них - O(N)

Но есть:

1) environment - тип переменных

2) join таблиц (разберем через занятие)

Хэш-таблицы в R

```r
ht <- new.env(hash = TRUE)

key <- "bar"
value <- "baz"

# first way
ht[[key]] <- value

# second way
ht$bar <- value

print(ls.str(ht))
```

```
## bar :  chr "baz"
```

Хэш-таблицы в R

```
ht <- new.env(hash = TRUE)
ht[["bar"]] <- "baz"

print(ht[["bar"]])
```

```
## [1] "baz"
```

```
print(ht$bar)
```

```
## [1] "baz"
```

Минусы

Не для этого environment изначально создавались.
Потому - работать с ними как с словарями не очень удобно

```
ht <- new.env(hash=TRUE)
ht["str"] <- 7
```

```
## Error in ht["str"] <- 7: object of type 'environment' is not subsettable
```

Минусы

Нельзя использовать в качестве ключа что-то, что не является строкой

```
ht <- new.env(hash=TRUE)
ht[[1]] <- 7
```

```
## Error in ht[[1]] <- 7: wrong args for environment subassignment
```

```
ht <- new.env(hash=TRUE)
ht[[c(1,2,3)]] <- 7
```

```
## Error in ht[[c(1, 2, 3)]] <- 7: wrong args for environment subassignment
```

Функции в R

```
dist_to_origin <- function(x){
    return ((x[1] * x[1] + x[2] * x[2]) ** 0.5)
}
```

Задали функцию, которая для вектора из двух элементов считает расстояние до точки (0, 0)

```
dist_to_origin <- function(x){
    (x[1] * x[1] + x[2] * x[2]) ** 0.5
}
```

То же самое, возвращается результат последнего вычисления

```
dist_to_origin <- function(x, ...){
    (x[1] * x[1] + x[2] * x[2]) ** 0.5
}
```

То же самое, ... означает - игнорируем этот аргумент (аргументы)

Функции в R

```
dist_to_origin <- function(x){
  (x[1] * x[1] + x[2] * x[2]) ** 0.5
}
print(dist_to_origin(c(1,2)))
```

```
## [1] 2.236068
```

```
print(dist_to_origin(c(1,2), 45, 98))
```

```
## Error in dist_to_origin(c(1, 2), 45, 98): unused arguments (45, 98)
```

```
dist_to_origin <- function(x, ...){
  (x[1] * x[1] + x[2] * x[2]) ** 0.5
}

print(dist_to_origin(c(1,2)))
```

```
## [1] 2.236068
```

```
print(dist_to_origin(c(1,2), 45, 98))
```

```
## [1] 2.236068
```

Ленивое вычисление

Функции в R вычисляют значения аргументов только тогда, когда аргументы понадобятся. Потому, приведенный ниже код не вызывает ошибки

```
fun <- function(a, b){
  if (a == 0){
    return(b)
  }else{
    return(a)
  }
}

print(fun(5, 10))
```

```
## [1] 5
```

```
print(fun(5))
```

```
## [1] 5
```

```
print(fun(0, 10))
```

```
## [1] 10
```

```
print(fun(0))
```

```
## Error in fun(0): argument "b" is missing, with no default
```

Ленивое вычисление

Функции в R вычисляют значения аргументов только тогда, когда аргументы понадобятся. Потому, приведенный ниже код не вызывает ошибки

```
f <- function(x){
  x
}

g <- function(x){
  10
}

f(log(-10))
```

```
## Warning in log(-10): NaNs produced
```

```
## [1] NaN
```

```
g(log(-10))
```

```
## [1] 10
```

Аргументы по-умолчанию

```
fun <- function(a, b=0){
  if (a == 0){
    return(b)
  }else{
    return(a)
  }
}
print(fun(5, 10))
```

```
## [1] 5
```

```
print(fun(5))
```

```
## [1] 5
```

```
print(fun(0, 10))
```

```
## [1] 10
```

```
print(fun(0))
```

```
## [1] 0
```

Аргументы по-умолчанию после ...

```
fun <- function(a, ..., b=10){
  if (a == 0){
    return(b)
  }else{
    return(a)
  }
}
print(fun(5, 10))
```

```
## [1] 5
```

```
print(fun(5))
```

```
## [1] 5
```

```
print(fun(0, 20))
```

```
## [1] 10
```

```
print(fun(0, b=20))
```

```
## [1] 20
```

Неизменяемость аргументов

```
fun <- function(a_vec){
    a_vec[2] <- 0
}
x <- c(1,2,3)
xl <- list(1, 2, 3)
fun(x)
print(x)
```

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

```
fun <- function(a_l){
    a_l[[2]] <- 0
}
xl <- list(1, 2, 3)
fun(xl)
print(xl)
```

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

Разрешение имен

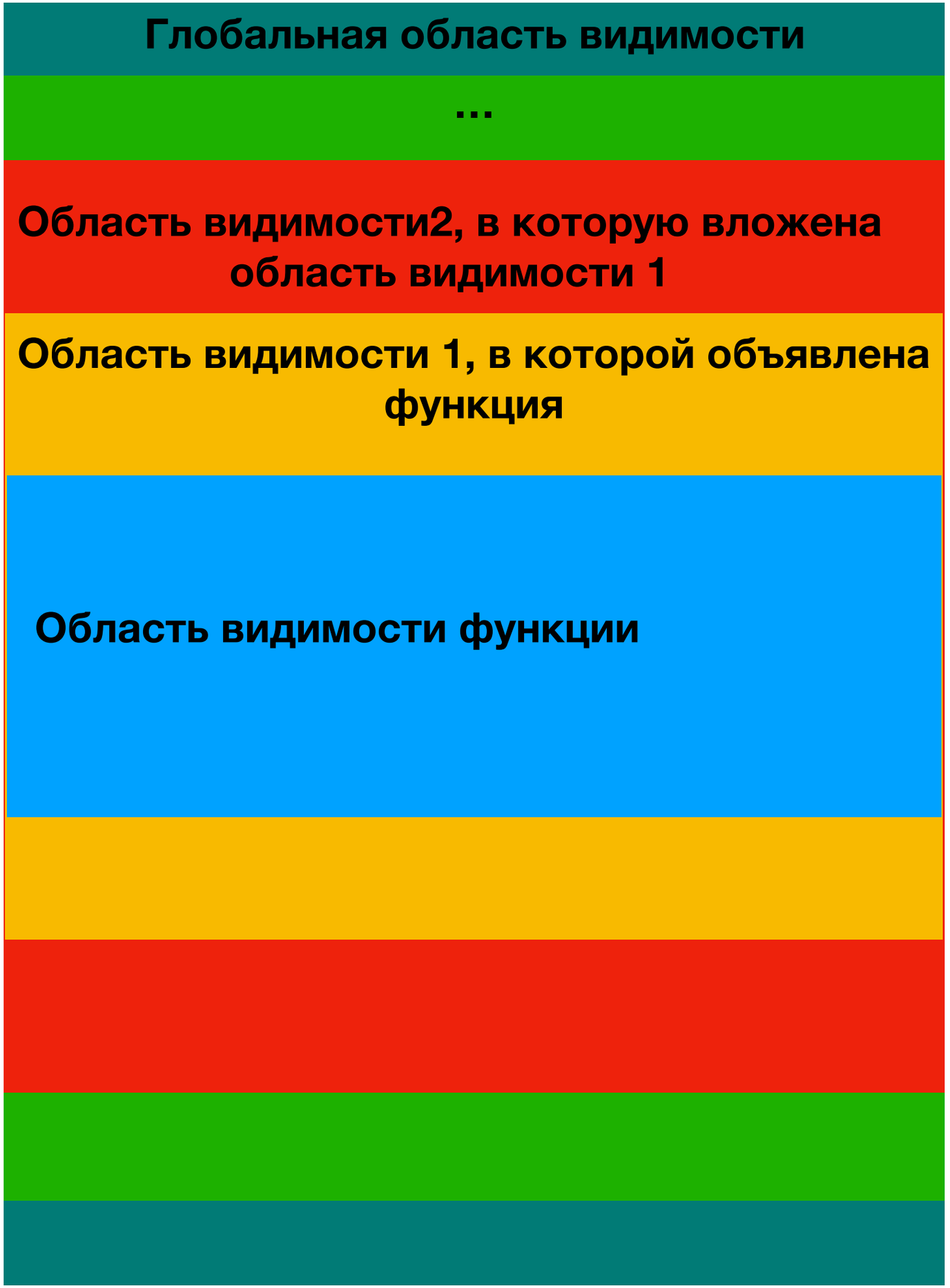
1) Сначала имя ищется в самой функции.

2) Если там оно не определено, то начинаем искать в окружении, где была определена функция

3) Если и там нет, то идем в родительское окружение

4) ….

5) Если нет и в самом “верхнем” окружении - кидается ошибка


Глобальная область видимости
...
Область видимости2, в которую вложена область видимости 1
Область видимости 1, в которой объявлена функция
Область видимости функции

Динамическое разрешение имен

Если переменная не была определена до определения функции, но была определена позже, то функция ее все равно увидит

```
fun <- function(a){
  if (a == 0){
    return(b)
  }else{
    return(a)
  }
}
fun(0)
```

```
## Error in fun(0): object 'b' not found
```

```
b <- 5
fun(0)
```

```
## [1] 5
```

Анонимные функции

```
function(x) { (x ** 3) %/% 2}
```

```
## function(x) { (x ** 3) %/% 2}
```

Анонимные функции

```
print(sapply(1:100, function(x){ (x ** 3) %/% 2}))
```

```
##   [1]     0     4    13    32    62   108   171   256   364   500
##  [11]   665   864  1098  1372  1687  2048  2456  2916  3429  4000
##  [21]  4630  5324  6083  6912  7812  8788  9841 10976 12194 13500
##  [31] 14895 16384 17968 19652 21437 23328 25326 27436 29659 32000
##  [41] 34460 37044 39753 42592 45562 48668 51911 55296 58824 62500
##  [51] 66325 70304 74438 78732 83187 87808 92596 97556 102689 108000
##  [61] 113490 119164 125023 131072 137312 143748 150381 157216 164254 171500
##  [71] 178955 186624 194508 202612 210937 219488 228266 237276 246519 256000
##  [81] 265720 275684 285893 296352 307062 318028 329251 340736 352484 364500
##  [91] 376785 389344 402178 415292 428687 442368 456336 470596 485149 500000
```

Анонимные функции

```
print(lapply(list(1:10, 1:100), function(x) {length(x)}))
```

```
## [[1]]
## [1] 10
##
## [[2]]
## [1] 100
```

Анонимные функции

```
print(mapply(function(x, y) { x * y}, 1:100, 1:100))
```

```
##   [1]     1     4     9    16    25    36    49    64    81   100   121   144
##  [13]   169   196   225   256   289   324   361   400   441   484   529   576
##  [25]   625   676   729   784   841   900   961  1024  1089  1156  1225  1296
##  [37]  1369  1444  1521  1600  1681  1764  1849  1936  2025  2116  2209  2304
##  [49]  2401  2500  2601  2704  2809  2916  3025  3136  3249  3364  3481  3600
##  [61]  3721  3844  3969  4096  4225  4356  4489  4624  4761  4900  5041  5184
##  [73]  5329  5476  5625  5776  5929  6084  6241  6400  6561  6724  6889  7056
##  [85]  7225  7396  7569  7744  7921  8100  8281  8464  8649  8836  9025  9216
##  [97]  9409  9604  9801 10000
```

Анонимные функции

```
sapply(1:10, function(x){
  sapply(1:10, function(y) { x %/% y})
})
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
##  [1,]    1    2    3    4    5    6    7    8    9    10
##  [2,]    0    1    1    2    2    3    3    4    4     5
##  [3,]    0    0    1    1    1    2    2    2    3     3
##  [4,]    0    0    0    1    1    1    1    2    2     2
##  [5,]    0    0    0    0    1    1    1    1    1     2
##  [6,]    0    0    0    0    0    1    1    1    1     1
##  [7,]    0    0    0    0    0    0    1    1    1     1
##  [8,]    0    0    0    0    0    0    0    1    1     1
##  [9,]    0    0    0    0    0    0    0    0    1     1
## [10,]    0    0    0    0    0    0    0    0    0     1
```

```
sapply(1:10, function(x){
  x %/% 1:10
})
```

Анонимные функции

```
sapply(1:5, function(mean){
  sapply(1:5, function(sd) {rnorm(1, mean, sd)})
})
```

```
##              [,1]      [,2]      [,3]      [,4]      [,5]
## [1,]  0.67748494  3.258096  1.804230  4.139071  4.298601
## [2,]  0.02905765  1.276234  2.490430  2.396537  5.391296
## [3,] -1.58607478  3.912907  5.676245  4.953701  7.840136
## [4,]  4.91132005 -1.519574 10.545251  5.431876 -1.284726
## [5,]  0.35354375  8.892712 -4.293606 10.257444  7.595371
```

R. Работа с таблицами

Tidyverse

Tidyverse

Packages Articles Learn Help Contribute

R packages for data science

The tidyverse is an opinionated collection of R packages designed for data science. All packages share an underlying design philosophy, grammar, and data structures.

Install the complete tidyverse with:

```
install.packages("tidyverse")
```

Плюсы

- Сравнительно легко освоить
- Очень удобен в использовании
- Общий синтаксис для всех пакетов
- Хорошо взаимодействует с ggplot (не входи в tidyverse) - пакетом для визуализации графики

Минусы

- Не весь R
- Синтаксис иногда меняется, в этом случае ломается все..
- Куча слоев абстракции - конкретная задача решается не напрямую, а с использованием кучи промежуточных шагов
- Как результат - можно получить абсолютно непонятный код, еще и работающий медленно

dplyr

Манипуляция данными в виде таблицы

Краткая инструкция по командам dplyr

https://github.com/rstudio/cheatsheets/blob/master/data-transformation.pdf

Установка:

```
install.packages("dplyr")
library(dplyr)
```

Примеры команд

select - выбрать какие-то колонки по имени

```
head(select(mtcars, mpg, cyl, disp))
```

```
##                   mpg cyl disp
## Mazda RX4          21.0   6  160
## Mazda RX4 Wag      21.0   6  160
## Datsun 710         22.8   4  108
## Hornet 4 Drive     21.4   6  258
## Hornet Sportabout  18.7   8  360
## Valiant            18.1   6  225
```

Имена колонок

Имя таблицы

Примеры команд

select - выбрать какие-то колонки по имени

```
head(select(mtcars, c("mpg", "cyl", "disp")))
```

```
##                   mpg cyl disp
## Mazda RX4         21.0   6  160
## Mazda RX4 Wag     21.0   6  160
## Datsun 710        22.8   4  108
## Hornet 4 Drive    21.4   6  258
## Hornet Sportabout 18.7   8  360
## Valiant           18.1   6  225
```

Имена колонок

Имя таблицы

Примеры команд

select - выбрать какие-то колонки по имени можно использовать условия для выбора колонок, например - starts_with позволит выбрать колонки, начинающиеся с определенного шаблона

```
selected <- select(iris, starts_with("Sepal"))
head(selected)
```

```
##   Sepal.Length Sepal.Width
## 1          5.1         3.5
## 2          4.9         3.0
## 3          4.7         3.2
## 4          4.6         3.1
## 5          5.0         3.6
## 6          5.4         3.9
```

Условие на колонки

Имя таблицы

Примеры команд

select - выбрать какие-то колонки по имени можно использовать условия для выбора колонок, например - starts_with позволит выбрать колонки, начинающиеся с определенного шаблона. Можно комбинировать

```
selected <- select(iris, Species, starts_with("Sepal"))
head(selected)
```

```
##   Species Sepal.Length Sepal.Width
## 1  setosa          5.1         3.5
## 2  setosa          4.9         3.0
## 3  setosa          4.7         3.2
## 4  setosa          4.6         3.1
## 5  setosa          5.0         3.6
## 6  setosa          5.4         3.9
```

Условие на колонки

Имя колонки

Имя таблицы

Примеры команд

pull- выбрать одну конкретную колонку и взять ее как вектор

```
selected <- pull(iris, Sepal.Length)
print(selected)
```

```
##   [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9 5.4 4.8 4.8
##  [18] 5.1 5.7 5.1 5.4 5.1 4.6 5.1 4.8 5.0 5.0 5.2 5.2 4.7
##  [35] 4.9 5.0 5.5 4.9 4.4 5.1 5.0 4.5 4.4 5.0 5.1 4.8 5.1
##  [52] 6.4 6.9 5.5 6.5 5.7 6.3 4.9 6.6 5.2 5.0 5.9 6.0 6.1
##  [69] 6.2 5.6 5.9 6.1 6.3 6.1 6.4 6.6 6.8 6.7 6.0 5.7 5.5
##  [86] 6.0 6.7 6.3 5.6 5.5 5.5 6.1 5.8 5.0 5.6 5.7 5.7 6.2
## [103] 7.1 6.3 6.5 7.6 4.9 7.3 6.7 7.2 6.5 6.4 6.8 5.7 5.8
## [120] 6.0 6.9 5.6 7.7 6.3 6.7 7.2 6.2 6.1 6.4 7.2 7.4 7.9
## [137] 6.3 6.4 6.0 6.9 6.7 6.9 5.8 6.8 6.7 6.7 6.3 6.5 6.2
```

Имя таблицы

Имя колонки

Примеры команд

filter - отфильтровать строки по условию

```
filtered <- filter(iris, Sepal.Length > 5)
head(filtered)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Wi
## 1          5.1         3.5          1.4
## 2          5.4         3.9          1.7
## 3          5.4         3.7          1.5
## 4          5.8         4.0          1.2
## 5          5.7         4.4          1.5
## 6          5.4         3.9          1.3
```

Имя таблицы

Условие

Примеры команд

slice - выбрать строки по позиции

```
sliced <- slice(iris, 1:10)
head(sliced)
```

```
##    Sepal.Length Sepal.Width Petal.Length Petal.Width Specie
## 1           5.1         3.5          1.4         0.2  setos
## 2           4.9         3.0          1.4         0.2  setos
## 3           4.7         3.2          1.3         0.2  setos
## 4           4.6         3.1          1.5         0.2  setos
## 5           5.0         3.6          1.4         0.2  setos
```

Имя таблицы

Позиции

Примеры команд

slice - выбрать строки по позиции

```
sliced <- slice(iris, (n()-10):n())
head(sliced)
```

Позиции, n()-число записей

Имя таблицы

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
## 1          6.9         3.1          5.4         2.1 virginica
## 2          6.7         3.1          5.6         2.4 virginica
## 3          6.9         3.1          5.1         2.3 virginica
## 4          5.8         2.7          5.1         1.9 virginica
## 5          6.8         3.2          5.9         2.3 virginica
```

Примеры команд

sample_n - Выбрать 10 случайных строк

```
sampled <- sample_n(iris, 10)
head(sampled)
```

Имя таблицы

число сэмплируемых объектов

```
##     Sepal.Length Sepal.Width Petal.Length Petal.
## 1            5.7         4.4          1.5
## 2            7.4         2.8          6.1
## 3            4.4         3.2          1.3
## 4            6.3         3.4          5.6
## 5            7.7         2.6          6.9
## 6            5.1         3.5          1.4
```

Примеры команд

arrange - отсортировать строки по колонкам

```
sorted <- arrange(iris, Sepal.Length)
head(sorted)
```

```
##    Sepal.Length Sepal.Width Petal.Length Pet
## 1           4.3         3.0          1.1
## 2           4.4         2.9          1.4
## 3           4.4         3.0          1.3
## 4           4.4         3.2          1.3
## 5           4.5         2.3          1.3
## 6           4.6         3.1          1.5
```

Имя таблицы

колонка,
по которой сортируем

Примеры команд

arrange - отсортировать строки по колонкам

```
sorted <- arrange(iris, desc(Sepal.Length))
head(sorted)
```

колонка, по которой сортируем в порядке убывания

Имя таблицы

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Wid
## 1          7.9         3.8          6.4          2
## 2          7.7         3.8          6.7          2
## 3          7.7         2.6          6.9          2
## 4          7.7         2.8          6.7          2
## 5          7.7         3.0          6.1          2
## 6          7.6         3.0          6.6          2
```

Примеры команд

arrange - отсортировать строки по колонкам

```
sorted <- arrange(iris, Sepal.Length, desc(Sepal.Width))
head(sorted)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1          4.3         3.0          1.1         0.1
## 2          4.4         3.2          1.3         0.2
## 3          4.4         3.0          1.3         0.2
## 4          4.4         2.9          1.4         0.2
## 5          4.5         2.3          1.3         0.3
## 6          4.6         3.6          1.0         0.2
```

Имя таблицы

колонки, по которой сортируем, сначала по первой, потом по второй

Примеры команд

add_row -добавить строку в таблицу

```
added <- add_row(iris,
                 Sepal.Length=1,
                 Sepal.Width=2,
                 Petal.Length=3,
                 Petal.Width=0.2,
                 Species='mine',
                 .before=T)
head(added)
```

Имя таблицы

Значения колонок

Вставлять в начало

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          1.0         2.0          3.0         0.2    mine
## 2          5.1         3.5          1.4         0.2  setosa
## 3          4.9         3.0          1.4         0.2  setosa
## 4          4.7         3.2          1.3         0.2  setosa
## 5          4.6         3.1          1.5         0.2  setosa
## 6          5.0         3.6          1.4         0.2  setosa
```

Примеры команд

add_column -добавить столбец в таблицу (функция из пакета tibble)

```
added <- add_column(iris, mine=1:nrow(iris))
head(added)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species mine
## 1          5.1         3.5          1.4         0.2  setosa    1
## 2          4.9         3.0          1.4         0.2  setosa    2
## 3          4.7         3.2          1.3         0.2  setosa    3
## 4          4.6         3.1          1.5         0.2  setosa    4
## 5          5.0         3.6          1.4         0.2  setosa    5
## 6          5.4         3.9          1.7         0.4  setosa    6
```

Значения Строк

Имя таблицы

Имя новой колонки

Примеры команд

summarise - подсчитать какое-то значение по всей колонке

```
summarise(iris, mean_sepal_length=mean(Sepal.Length) )
```

```
##   mean_sepal_length
## 1          5.843333
```

Имя таблицы

Имя новой колонки

Примеры команд

summarise - подсчитать какое-то значение по всей колонке

Имя новой колонки

```
summarise(iris,
          mean_sepal_length=mean(Sepal.Length),
          sd_sepal_length=sd(Sepal.Length))
```

```
##   mean_sepal_length sd_sepal_length
## 1          5.843333       0.8280661
```

Имя таблицы

Примеры команд

summarise - подсчитать какое-то значение по всей колонке

Имя новой колонки

```
summarise(iris,
          mean_sepal_length=quantile(Sepal.Length, 0.55))
```

```
##   mean_sepal_length
## 1                 5.9
```

Имя таблицы

Примеры команд

count - подсчитать число строк, в которых значение переменной равно тому-то

```
count(iris, Sepal.Length)
```

Имя колонки

Имя таблицы

```
## # A tibble: 35 x 2
##    Sepal.Length     n
##           <dbl> <int>
##  1          4.3     1
##  2          4.4     3
##  3          4.5     1
##  4          4.6     4
##  5          4.7     2
```

Примеры команд

count - подсчитать число строк, в которых значение переменной равно тому-то

```
count(iris, Sepal.Length, Sepal.Width)
```

Имя колонки

Имя таблицы

```
## # A tibble: 117 x 3
##    Sepal.Length Sepal.Width     n
##           <dbl>       <dbl> <int>
##  1          4.3           3     1
##  2          4.4           2.9   1
##  3          4.4           3     1
##  4          4.4           3.2   1
##  5          4.5           2.3   1
##  6          4.6           3.1   1
##  7          4.6           3.2   1
##  8          4.6           3.4   1
##  9          4.6           3.6   1
## 10          4.7           3.2   2
```

Примеры команд

mutate - подсчитать новую колонку

```
mutated <- mutate(iris, inv_length=1/Sepal.Length)
head(mutated)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Speci
## 1          5.1         3.5          1.4         0.2  seto
## 2          4.9         3.0          1.4         0.2  seto
## 3          4.7         3.2          1.3         0.2  seto
## 4          4.6         3.1          1.5         0.2  seto
## 5          5.0         3.6          1.4         0.2  seto
## 6          5.4         3.9          1.7         0.4  seto
```

Имя таблицы

Имя колонки

Примеры команд

mutate - подсчитать новую колонку

```
mutated <- mutate(iris,
                  inv_length=1/Sepal.Length,
                   inv_width=1/Sepal.Width)

head(mutated)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
##   inv_width
## 1 0.2857143
## 2 0.3333333
```

Имя таблицы

Имя колонки

Примеры команд

transmute - подсчитать новую колонку, убрать остальные

```
transmuted <- transmute(iris, inv_length=1/Sepal.Length)
head(transmuted)
```

```
##    inv_length
## 1   0.1960784
## 2   0.2040816
## 3   0.2127660
## 4   0.2173913
## 5   0.2000000
## 6   0.1851852
```

Имя колонки

Имя таблицы

Примеры команд

transmute - подсчитать новую колонку, убрать остальные

```
transmuted <- transmute(iris,
                        inv_length=1/Sepal.Length,
                        inv_width=1/Sepal.Width)

head(transmuted)
```

```
##    inv_length inv_width
## 1   0.1960784 0.2857143
## 2   0.2040816 0.3333333
## 3   0.2127660 0.3125000
## 4   0.2173913 0.3225806
## 5   0.2000000 0.2777778
## 6   0.1851852 0.2564103
```

Имя колонки

Имя таблицы

Примеры команд

rename - переименовать колонку

```
renamed <- rename(iris,
                  Length=Sepal.Length)

head(renamed)
```

Новое имя колонки

Старое имя колонки

```
##   Length Sepal.Width Petal.Length Petal.Width Species
## 1    5.1         3.5          1.4         0.2  setosa
## 2    4.9         3.0          1.4         0.2  setosa
## 3    4.7         3.2          1.3         0.2  setosa
## 4    4.6         3.1          1.5         0.2  setosa
## 5    5.0         3.6          1.4         0.2  setosa
## 6    5.4         3.9          1.7         0.4  setosa
```

Имя таблицы

Pipe %>%

Имя таблицы

Позволяет направить таблицу,
полученную одной командой, в другую команду в качестве
первого аргумента

```
starwars %>%
  mutate(bmi = mass / ((height / 100)  ^ 2)) %>%
  select(name:mass, bmi)
```

Добавляем индекс массы тела

Выбираем колонки от name до mass

```
## # A tibble: 87 x 4
##    name               height  mass   bmi
##    <chr>               <int> <dbl> <dbl>
##  1 Luke Skywalker        172    77  26.0
##  2 C-3PO                 167    75  26.9
##  3 R2-D2                  96    32  34.7
```

group_by

```
starwars %>%
  group_by(species) %>%
    count()
```

```
## # A tibble: 38 x 2
## # Groups:   species [38]
##    species        n
##    <chr>      <int>
##  1 <NA>           5
##  2 Aleena         1
##  3 Besalisk       1
```

Позволяет сгруппировать таблицу (фактически - разбить ее на под-таблицы) по колонке/группе колонок

Pipe %>%

```
starwars %>%
  group_by(species) %>%
  summarise(
    n = n(),
    mass = mean(mass, na.rm = TRUE)
  ) %>%
  filter(n > 1,
         mass > 50)
```

Группируем по species

Посчитаем, сколько какого вида и среднюю массу

Отберем случаи, когда представителей виде было больше 1 и средняя масса вида больше 50

```
## # A tibble: 8 x 3
##   species       n  mass
##   <chr>     <int> <dbl>
## 1 Droid         5  69.8
## 2 Gungan        3  74
```

Можно присваивать

```
species_mass_filtered <- starwars %>%
  group_by(species) %>%
  summarise(
    n = n(),
    mass = mean(mass, na.rm = TRUE)
) %>%
filter(n > 1,
       mass > 50)
```

Аналог dplyr

data.tables

Функционал схожий
Работает в разы быстрее

Синтаксис на первый взгляд менее понятен

```
DT[i, j, by]

##   R:                 i                 j        by
## SQL:  where | order by   select | update  group by
```

data.tables

Синтаксис на первый взгляд менее понятен

Да и на второй.

Повторение задачи с starwars

```
starwars <- as.data.table(starwars)
starwars[, .(num=.N, mass=mean(mass, na.rm=T)), species][num > 1 & mass > 50, ]
```

```
##      species num      mass
## 1:     Human  35  82.78182
## 2:     Droid   5  69.75000
## 3:   Wookiee   2 124.00000
## 4:    Gungan   3  74.00000
## 5:    Zabrak   2  80.00000
## 6:   Twi'lek   2  55.00000
## 7: Mirialan   2  53.10000
## 8: Kaminoan   2  88.00000
```

Зато он быстрее

data.tables

- Функция fread из этого пакета - очень быстрое чтение таблиц. Можно использовать только ее.