

Лабораторная работа №1 (4 часа)

Введение в язык R

Цель работы: Ознакомление с синтаксисом языка **R** для анализа данных.

1.1 Общие сведения

Язык программирования **R** является диалектом (реализацией) языка S, который был разработан как эффективное и удобное средство для работы с данными, моделирования и визуализации. **R** развивается в рамках open-source-проекта и доступен для различных платформ: Windows, MacOS и UNIX/UNIX-подобных систем (включая MAC OS, FreeBSD и Linux).

Полезные ссылки:

Краткий справочник часто используемых функций:

- <http://cran.r-project.org/doc/contrib/Short-refcard.pdf> (англ.),
- <http://aakinshin.net/ru/blog/r/functions/> (рус.)

Популярная свободная среда разработки – <http://www.rstudio.com>

1.2. Основы работы в системе R

Основным способом работы пользователя в системе **R** является использование интерактивного режима работы в консоли интерпретатора, либо написание программ-скриптов для их дальнейшего исполнения в системе. Как правило, скрипты сохраняются в файлах, имеющих расширение .R. Вызов скрипта на исполнение из консоли интерпретатора языка осуществляется функцией `source("path/to/file.R")`. При этом можно указывать относительные пути, отправной точкой которых является рабочая директория, узнать которую можно с помощью функции `getwd()`, а изменить с помощью `setwd("new/working/directory")`.

Все вычисления производятся в некотором окружении (environment), которое определяет связь между именами и значениями: между именем переменной и ее значением, между именем функции и ее реализацией. Данное связывание производится путем объявления и инициализации переменных с помощью оператора присваивания (= или <-). **Дополнительного указания типа переменной при ее создании не требуется.**

```
1 | > x = 7
2 | > x
3 | [1] 7
4 | > y <- 1.5
5 | > y
6 | [1] 1.5
```

Для вывода результата на экран в скриптах необходимо использовать функции *print(x)*, *cat(x)* или *sprintf(fmt,...)*. Функция *sprintf* служит для форматированного вывода и имеет такой же формат, как и аналогичная функция в языке C. Для перенаправления вывода в файл используется функция *sink(file)*. Восстановление вывода в консоль производится вызовом данной функции без аргументов.

Узнать все доступные в текущем окружении имена можно с помощью функции *ls()*. Удалить некоторую переменную *x* можно с помощью функции *rm(x)*, полностью очистить текущее окружение можно следующим способом *rm(list= ls())*.

R – ООП язык функционального программирования

Любая операция в R это функция. Каждая команда или выражение в языке **R** возвращает некоторый результат. Выражение может быть частью другого выражения. В частности, возможны множественные присваивания. Выражения можно объединять в блоки. Блок – это несколько выражений, заключенных в фигурные скобки. Сами выражения, как обычно, разделяются точкой с запятой или символом перехода на новую строку. Результатом выполнения всего блока будет результат последнего из выражений, составляющих блок.

Для получения справки по той или иной функции можно выполнить команду *help("function_name")*.

Далее приводится краткая информация по существующим типам (структурам данных) языка R, основам программирования на данном языке, работе с пакетами расширений и графикой.

1.3. Базовые типы данных

Все типы данных можно представить двумя группами: **контейнеры** и **первичные** типы данных. Любая переменная в языке R может принимать только значение типа контейнера, элементами могут уже являться значения первичных типов данных.

Первичные типы:

1. Логические (logical);
2. Целочисленные (integer);
3. Вещественные числа (numeric);
4. Комплексные числа (complex);
5. Текстовые (character);
6. Списки (list).

Контейнеры:

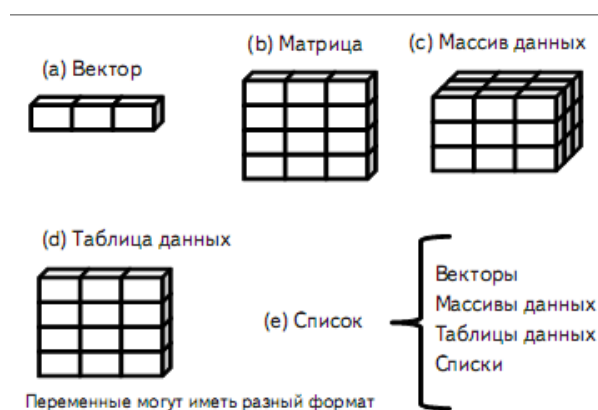


Рис. 1.1. Контейнерные типы данных в R

R — это объектно-ориентированный язык программирования. **Classes** (классы) реализуются как **objects** (объекты) соответствующего типа данных (то есть класс — это **тип данных для переменной**, а сама переменная этого типа является объектом — экземпляром класса). **Classes** (классы) присоединяются к объектам как **attribute** (атрибуты). Все объекты в **R** имеют **свой класс, тип и размерность**.

```
> class(object)
> typeof(object)
> dim(object)
```

Таблица 1. 1. Возможные типы данных в R

Объект	Возможные типы данных	Использование в объекте нескольких типов данных
vector	числовой, символьный, комплексный, логический	Нет
factor	числовой, символьный	Нет
array	числовой, символьный, комплексный, логический	Нет
matrix	числовой, символьный, комплексный, логический	Нет
data.frame	числовой, символьный, комплексный, логический	Да
Ts	числовой, символьный, комплексный, логический	Да
List	числовой, символьный, комплексный, логический, функция, выражение, формула	Да

Существует несколько присущих только R терминов, которые приводят в замешательство новых пользователей. В R **объектом (object)** называется все, что может быть представлено в виде переменных, включая константы, разные типы

данных, функции и даже диаграммы. У объектов есть вид (определяет, в каком виде объект хранится в памяти) и класс (который указывает общим функциям типа `print()`, как с ним обращаться).

Таблица данных (*data frame*) – это тип структуры данных в R, аналогичный тому виду, в котором хранятся данные в обычных статистических программах (например, в SAS, SPSS и STATA). Столбцы – это переменные, а строки – это наблюдения. В одной таблице данных могут содержаться переменные разных типов (например, числовые и текстовые). Таблицы данных – это основной тип структуры данных.

Факторы – это номинальные или порядковые переменные. В R они хранятся и обрабатываются особым образом. Вы узнаете больше о факторах далее из раздела 2.3.8.

Ts–набор данных временного ряда, содержащий дополнительные атрибуты, такие как: частота и дата.

1.3.1. Вектор

Векторы²⁾ являются основополагающей структурой данных языка **R**, представляющей собой некоторое количество однотипных элементов, непрерывно расположенных в памяти.

²⁾В терминологии языка программирования **R** различают *атомарные* (*atomic*) и *общие* (*generic*) векторы. Последние могут содержать *элементы различных типов*. Так, списки являются векторами, но не атомарными. Здесь, говоря о векторах, мы будем подразумевать атомарные.

В **R** есть 6 базовых типов векторов (или, что тоже самое, их элементов): **логический (logical), целочисленный (integer), вещественный (double), комплексный (complex), символьный (character) и двоичные данные (raw)**. Любая константа и переменная одного из перечисленных выше типов также является вектором длины 1. Для определения длины вектора служит функция ***length(x)***.

```
1 > length(7.8)
2 [1] 1
3 > length("text")
4 [1] 1
```

К числовым векторам можно применять арифметические операции (+, -, *, /, ^), которые будут выполнены поэлементно. Если длины векторов различны, то вектор меньшей длины будет повторен нужное количество раз и обрезан до нужного размера. К целочисленным векторам аналогично применимы операции получения целой части %/% и остатка %% от деления. Аналогичным образом векторы можно сравнивать с помощью операций ==, !=, <, <=, >, >=, результатом применения которых, являются логические векторы. К логическим векторам, в

свою очередь, можно применять логические операции & (конъюнкция), | (дизъюнкция), ! (отрицание). Элементы векторов кроме обычных значений, обусловленных их типом, могут принимать специальные значения: NA (недоступное, пропущенное значение), NaN (не число), Inf (бесконечность), -Inf (минус бесконечность).

Функции создания векторов

Приведем описание некоторых базовых функций для создания векторов:

integer(n), *double(n)*, *numeric(n)*, *complex(n)*, *logical(n)*, *character(n)*, *raw(n)*. создают векторы длины *n* соответствующего типа, инициализированные значениями по умолчанию. *numeric(n)* создает вектор чисел с плавающей запятой, т.е. типа *double*.

```
1 > integer(10)
2 [1] 0 0 0 0 0 0 0 0 0 0
3 > logical(5)
4 [1] FALSE FALSE FALSE FALSE FALSE
```

c(...). Объединяет (конкатенирует) несколько векторов в один.

```
1 > c(1, 2, 3)
2 [1] 1 2 3
3 > c(c(1, 2), 3)
4 [1] 1 2 3
5 > length(c(1, 2, 3))
6 [1] 3
```

Оператор: создает вектор, состоящий из последовательных целых чисел, границы интервала при этом указываются следующий образом:

<начальное_значение>:<конечное_значение>.

```
1 > 1:5
2 [1] 1 2 3 4 5
```

seq(from, to, by = d), *seq(from, to, length = n)* – Разбивает отрезок [from, to] на части с шагом *d*, либо на (*n* - 1) равных частей.

```
1 > seq(0, 1, by = 0.1)
2 [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
3 > seq(0, 1, len = 5)
4 [1] 0.00 0.25 0.50 0.75 1.00
```

rep(x, times). Конкатенирует *times* копий вектора *x*.

```
1 > rep(c(TRUE, FALSE), 3)
2 [1] TRUE FALSE TRUE FALSE TRUE FALSE
```

Оператор `[` создает вектор, состоящий из заданных элементов исходного вектора. Таким образом, `a[idx]` представляет собой новый вектор, содежащий элементы вектора `a`, указанные в `idx`. При этом вектор `idx` может состоять из положительных целых чисел и содержать в себе индексы элементов в `a` (индексация начинается с 1), которые нужно включить в новый вектор, либо состоять из отрицательных целых чисел и содержать в себе индексы элементов в `a`, которые нужно исключить из исходного вектора, либо быть логической маской.

```
1 > (10:20)[c(1, 3, 5)]
2 [1] 10 12 14
3 > (10:20)[-c(1, 3, 5)]
4 [1] 11 13 15 16 17 18 19 20
```

3)Здесь и далее приводится не полное описание аргументов функций. Подробное описание может быть найдено в документации.

sample(x, n, replace = FALSE, prob = NULL). Создает вектор, являющийся случайной выборкой *n* значений из вектора *x* с повторением или без, в зависимости от значения параметра ***replace***. Распределение вероятностей можно задать с помощью параметра ***prob***, по умолчанию все значения равновероятны.

```
1 > sample(1:10, 5, replace = FALSE)
2 [1] 5 8 2 1 9
3 > sample(1:10, 5, replace = TRUE)
4 [1] 10 2 6 3 10
```

Вектор можно создать также при помощи функции **`scan()`**, которая "считывает" последовательно вводимые с клавиатуры значения:

```
X <- scan()
1: 2.9 # после каждого нового значения нажать клавишу "Ввод"
2: 3.1
3: 3.4
4: 3.4
5: 3.7
6: 3.7
7: 2.8
8: 2.5
9: # выполнение команды scan завершают введением пустой строки
Read 8 items # программа сообщает о считывании 8 значений
X
[1] 2.9 3.1 3.4 3.4 3.7 3.7 2.8 2.5
```

Имена

Вы можете именовать вектор 3-мя способами:

- При создании: `x <- c(a = 1, b = 2, c = 3);`

- Изменяя существующий вектор: `x <- 1:3; names(x) <- c("a", "b", "c")`.

или: `x <- 1:3; names(x)[[1]] <- c("a");`

- Создавая модифицированную копию вектора: `x <- setNames(1:3, c("a", "b", "c"))`.

Имена не обязаны быть уникальными. Однако при замене содержимого элементов вектора ([subsetting](#)) удобнее использовать уникальные имена.

1.3.2. Математические функции

Приведем описание некоторых математических функций, применяемых к векторам:

- `sin(x)`, `cos(x)`, `tan(x)`, `asin(x)`, `acos(x)`, `atan(x)`, `atan2(x)` – Тригонометрические функции;
- `exp(x)`, `log(x)`, `log10(x)`, `log(x, base)` – Экспонента и логарифм;
- `max(x)`, `min(x)` – Поиск максимального/минимального значения;
- `range(x)` – Поиск минимума и максимума, т.е. `c(min(x), max(x))`;
- `sum(x)` – Сумма элементов вектора;
- `prod(x)` – Произведение элементов вектора;
- `mean(x)` – Среднее арифметическое элементов вектора;
- `median(x)` – Медиана (средний по величине элемент) вектора;
- `quantile(x, probs = seq(0, 1, 0.25))` – Подсчет выборочных квантилей указанных уровней;
- `var(x)` – Несмещенная оценка дисперсии;
- `sd(x)` – Оценка стандартного (среднеквадратичного) отклонения.

1.3.3. Вероятностные распределения

Приведем описание некоторых функций для генерации выборок реализаций случайных величин, распределенных по определенным законам. Все перечисленные ниже функции принимают в качестве своего первого параметра объем требуемой выборки (размер вектора, получаемого на выходе функции).

- `rnorm(n, mean = 0, sd = 1)` – Выборка из нормального распределения;
- `runif(n, min = 0, max = 1)` – Выборка из равномерного распределения;
- `rexp(n, rate = 1)` – Выборка из экспоненциального распределения;
- `rt(n, df)` – Выборка из распределения Стьюдента;
- `rf(n, df1, df2)` – Выборка из распределения Фишера;

- `rchisq(n, df)` – Выборка из распределения Пирсона;
- `rbinom(n, size, prob)` – Выборка из биномиального распределения;
- `rgeom(n, prob)` – Выборка из геометрического распределения;
- `rhyper(nn, m, n, k)` – Выборка из гипергеометрического распределения;
- `rlogis(n, location = 0, scale = 1)` – Выборка из логистического распределения;
- `rlnorm(n, meanlog = 0, sdlog = 1)` – Выборка из логнормального распределения;
- `rnbinom(n, size, prob)` – Выборка из отрицательного биномиального распределения.

1.3.4. Другие функции

Приведем описание еще нескольких полезных функций для работы с векторами:

- `sort(x, decreasing = FALSE)`. Создает вектор, состоящий из элементов вектора `x`, отсортированных в порядке убывания (`decreasing = TRUE`) или возрастания (`decreasing = FALSE`);
- `order(x, decreasing = FALSE)`. Создает вектор-перестановку элементов вектора `x` для его упорядочивания по возрастанию или убыванию. Сортировка значений вектора `x` может быть осуществлена следующим образом: `x[order(x)]`;
- `round(x)`, `trunc(x)`, `floor(x)`, `ceiling(x)`. Функции округления векторов с числами с плавающей запятой;
- `is.integer(x)`, `is.double(x)`, `is.numeric(x)`, `is.complex(x)`, `is.logical(x)`, `is.character(x)`, `is.raw(x)`. Функции для проверки типа вектора, возвращают логическое значение;
- `as.integer(x)`, `as.double(x)`, `as.numeric(x)`, `as.complex(x)`, `as.logical(x)`, `as.character(x)`, `as.raw(x)`. Функции конвертирования типов векторов;
- `is.na(x)`, `is.nan(x)`, `is.infinite(x)`. Функции для проверки являются ли элементы вектора равными NA, NaN, Inf/-Inf;
- `paste(..., sep = "", collapse = NULL)`. Функция для конкатенации векторов, сконвертированных в символьные. Данная функция конвертирует все, переданные в нее векторы в символьные, после чего поэлементно конкатенирует их, используя разделитель `sep`. Если указан параметр `collapse`, то все элементы полученного вектора также объединяются в одну строку, используя разделитель `collapse`:

```

1 > paste(c("a", "b", "c"), 1:3, sep = "-")
2 [1] "a-1" "b-2" "c-3"
3 > paste(c("a", "b", "c"), 1:3, sep = "-", collapse = "/")
4 [1] "a-1/b-2/c-3"
5 > paste(c("a", "b", "c"), collapse = "")
6 [1] "abc"
```


1.3.5. Матрицы

В то время как векторы представляют одномерную последовательность данных базовых типов, зачастую удобнее оперировать многомерными данными. Так матрицы в языке *R* представляют собой двумерные векторы. Для преобразования вектора в матрицу можно присвоить вектору атрибут размерности с помощью функции *dim(x)*. Данная функция принимает в качестве аргумента вектор, матрицу или массив, возвращая вектор размерностей. Вектор может иметь либо пустой вектор размерностей, либо вектор размерностей длины один, содержа в себе длину вектора. Матрица должна иметь вектор размерностей длины 2, первая компонента которого обозначает количество строк, вторая – столбцов. Данные в матрице хранятся по столбцам.

```
1 > x = 1:16
2 > dim(x)
3 NULL
4 > dim(x) = c(16)
5 > x
6 [1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
7 > dim(x)
8 [1] 16
9 > dim(x) = c(2, 8)      # первый способ создания матрицы
10 > x
11      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
12 [1,] 1 3 5 7 9 11 13 15
13 [2,] 2 4 6 8 10 12 14 16
```

Другим способом создания матриц является использование функции *matrix(data = NA, nrow = 1, ncol = 1, byrow = FALSE)*. Параметр *data* принимает вектор, элементами которого будет проинициализированная созданная матрица. Параметры *nrow* и *ncol* определяют количество строк и столбцов соответственно, *byrow* определяет, будет ли заполнена матрица элементами из *data* по строкам или по столбцам.

```
1 > x = 1:16
2 > y = matrix(data = x, nrow = 2, ncol = 8)
3 > x
4 [1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
5 > y
6      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
7 [1,] 1 3 5 7 9 11 13 15
8 [2,] 2 4 6 8 10 12 14 16
9 > dim(y)
10 [1] 2 8
11 > matrix(1:3, nrow = 2, ncol = 3, byrow = FALSE)
12      [,1] [,2] [,3]
13 [1,] 1 3 2
14 [2,] 2 1 3
```

```

15 > matrix (1:3, nrow = 2, ncol = 3, byrow = TRUE)
16      [,1] [,2] [,3]
17 [1,]  1   2   3
18 [2,]  1   2   3

```

Узнать количество строк и столбцов матрицы можно, получив весь вектор размерностей с помощью функции **dim**, либо используя функции **nrow(x)** и **ncol(x)**. Доступ к элементам матрицы происходит по индексу. **A[i, j]** ссылается на элемент *i*-й строки и *j*-го столбца матрицы **A**. На месте индексов *i* и *j* могут стоять векторы. Интерпретация такая же, как и для векторов.

Если, например, **i** и **j** это векторы с целыми положительными элементами, то **A[i, j]** есть подматрица матрицы **A**, образованная элементами, стоящими на пересечении строк с номерами из **i** и столбцов с номерами из **j**. Если, **i** и **j** это векторы с целыми отрицательными элементами, то **A[i, j]** есть подматрица, полученная из **A** *вычеркиванием* соответствующих строк и столбцов. Векторы **i** и **j** могут быть логическими или символьными. В последнем случае строкам и столбцам матрицы должны быть присвоены имена функций **rownames(x)** и **colnames(x)**. **A[i,]** эквивалентно **A[i, 1:ncol(A)]**, а **A[, j]** эквивалентно **A[1:nrow(A), j]**. Возможен доступ к элементам матрицы с помощью одного индекса. Например, **A[5]** означает 5-й элемент матрицы **A**, если считать, что элементы перенумерованы по столбцам. Если **i** вектор, то **A[i]** означает выборку соответствующих элементов и т.д.

```

1 > m = matrix (1:6, nrow = 2, ncol = 3)
2 > m
3 [,1] [,2] [,3]
4 [1,]  1   3   5
5 [2,]  2   4   6
6 > m[1, 3]
7 [1] 5
8 > m[, 3]
9 [1] 5 6
10 > m[, c(1, 3)]
11 [,1] [,2]
12 [1,]  1   5
13 [2,]  2   6
14 > m[-1,]
15 [1] 2 4 6
16 > A = matrix (c(23, 31, 58, 16), nrow = 2)
17 > rownames(A) <- c("petal", "sepal")
18 > colnames(A) <- c("length", "width")
19 > A
20 length width
21 petal      23   58
22 sepal      31   16
23 > A["petal", "length"]
24 [1] 23

```

Для того, чтобы объединить несколько матриц в одну, используются функции **rbind(...)** (приписывает матрицы друг к другу снизу) и **cbind(...)** (приписывает матрицы друг к другу справа).

```

1 > A = matrix (1:4, nrow = 2, ncol = 2)
2 > B = matrix (5:8, nrow = 2, ncol = 2)
3 > cbind(A, B)
4      [,1] [,2] [,3] [,4]
5 [1,]    1    3    5    7
6 [2,]    2    4    6    8
7 > rbind(A, B)
8      [,1] [,2]
9 [1,]    1    3
10 [2,]    2    4
11 [3,]    5    7
12 [4,]    6    8

```

К матрицам можно применять арифметические операции, однако, они будут осуществляться покомпонентно. Для осуществления матричного умножения применяется оператор `%*%`.

Рассмотрим еще несколько функций, часто используемых при работе с матрицами.

t(A). Транспонирует матрицу.

diag(A). Возвращает вектор диагональных элементов матрицы, либо создает диагональную матрицу с указанными значениями диагонали.

```

1 > m = matrix (1:6, nrow = 2, ncol = 3)
2 > m
3      [,1] [,2] [,3]
4 [1,]    1    3    5
5 [2,]    2    4    6
6 > diag(m)
7 [1] 1 4
8 > diag(m) = c(11, 14)
9 > m
10     [,1] [,2] [,3]
11 [1,]   11    3    5
12 [2,]    2   14    6
13 > a = diag(1:3, nrow = 3, ncol = 4)
14 > a
15     [,1] [,2] [,3] [,4]
16 [1,]    1    0    0    0
17 [2,]    0    2    0    0
18 [3,]    0    0    3    0

```

- ***det(A)***. Вычисляет определитель матрицы;
- ***solve(A, b)***. Решает систему линейных уравнений $Ax = b$ с квадратной и невырожденной матрицей A. При этом ***solve(A)*** может применяться для отыскания обратной матрицы;
- ***eigen(A)***. Вычисляет собственные числа и векторы. Возвращает список, содержащий собственные числа и матрицу собственных векторов;
- ***chol(A)***. Выполняет разложение Холецкого для симметричной положительно определенной матрицы;
- ***qr(A)***. Выполняет QR-разложение матрицы. Возвращает список, содержащий значения, полученные в результате разложения матрицы;

- ***qr.solve(A, b, tol = 1e-7)***. Решает систему линейных уравнений через QR-разложение;
- ***svd(A)***. Выполняет сингулярное разложение матрицы. Возвращает список, содержащий полученные в результате разложения матрицы.

1.3.6. Массив

Дальнейшим развитием идеи многомерности данных в языке **R** являются массивы, которые могут иметь *произвольное количество размерностей*. Работа с массивами почти идентично работе с матрицами. Создать массив можно либо изменив вектор размерностей вектора или матрицы с помощью функции ***dim(x)***, либо используя функцию ***array(data = NA, dim = length(data))***.

```

1 > A = array(1:24, c(2, 3, 4))
2 > A
3 , , 1
4
5      [,1] [,2] [,3]
6 [1,]    1    3    5
7 [2,]    2    4    6
8
9 , , 2
10
11     [,1] [,2] [,3]
12 [1,]    7    9   11
13 [2,]    8   10   12
14
15 , , 3
16
17     [,1] [,2] [,3]
18 [1,]   13   15   17
19 [2,]   14   16   18
20
21 , , 4
22
23     [,1] [,2] [,3]
24 [1,]   19   21   23
25 [2,]   20   22   24
26
27 > x = 1:24
28 > dim(x) = c(2, 3, 4)
29 > x
30 , , 1
31
32     [,1] [,2] [,3]
33 [1,]    1    3    5
34 [2,]    2    4    6
35
36 , , 2
37
```

```

38      [,1] [,2] [,3]
39 [1,]  7   9  11
40 [2,]  8  10  12
41
42 , , 3
43
44      [,1] [,2] [,3]
45 [1,] 13  15  17
46 [2,] 14  16  18
47
48 , , 4
49      [,1] [,2] [,3]
50 [1,] 19  21  23
51 [2,] 20  22  24

```

Для проверки является ли некоторая переменная вектором (не атомарным, а общим), матрицей или массивом используются функции *is.vector(x)*, *is.matrix(x)*, *is.array(x)*.

1.3.7. Список

Как уже отмечалось выше, список является общим (generic) вектором, т.е. его элементы могут иметь *различный тип*. Элементами списка могут быть, в том числе, векторы и другие списки. Для создания списка служит функция *list(...)*, принимающая на вход произвольное количество объектов, которые требуется объединить в список.

```

1 > L1 = list(1:5, c("a", "b"))
2 > L1
3 [[1]]
4 [1] 1 2 3 4 5
5
6 [[2]]
7 [1] "a" "b"
8
9 > L2 = list(rep(c(TRUE, FALSE), 3), L1)
10 > L2
11 [[1]]
12 [1] TRUE FALSE      TRUE FALSE  TRUEFALSE
13
14 [[2]]
15 [[2]][[1]]
16 [1] 1 2 3 4 5
17
18 [[2]][[2]]
19 [1] "a" "b"

```

Для добавления новых элементов в список может служить функция конкатенации списков *c(...)*. Применение к спискам способов индексирования, свойственных векторам, приведет к созданию нового списка, содержащего только выбранные элементы, даже если такой элемент всего один. Для доступа к

элементам списка используется индексирование с помощью двойных квадратных скобок.

```
1 > x = list(1:5, c("a", "b", "c"), rep(c(FALSE, TRUE), 3), seq(0, 1, length = 11))
2 > x
3 [[1]]
4 [1] 1 2 3 4 5
5
6 [[2]]
7 [1] "a" "b" "c"
8
9 [[3]]
10 [1] FALSE TRUE FALSE TRUE FALSE TRUE
11
12 [[4]]
13 [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
14
15 > x[c(1, 3)]
16 [[1]]
17 [1] 1 2 3 4 5
18
19 [[2]]
20 [1] FALSE TRUE FALSE TRUE FALSE TRUE
21
22 > x[2]
23 [[1]]
24 [1] "a" "b" "c"
25
26 > x[[2]]
27 [1] "a" "b" "c"
```

Также, с помощью функции *names(x)* элементам списка можно присваивать имена, а затем использовать их в качестве индексов, указывая в квадратных скобках (одинарных или двойных) после имени списка, либо используя оператор \$.

```
1 > x = list(1:5, c("a", "b", "c"), rep(c(FALSE, TRUE), 3), seq(0, 1, length = 11))
2 > names(x) = c("integers", "letters", "booleans", "doubles")
3 > x
4 $integers
5 [1] 1 2 3 4 5
6
7 $letter
8 [1] "a" "b" "c"
9
10 $booleans
11 [1] FALSE TRUE FALSE TRUE FALSE TRUE
12
13 $doubles
14 [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
15
16 > x[c("integers", "doubles")]
17 $integers
18 [1] 1 2 3 4 5
19
20 $doubles
```

```

21 [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
22
23 > x["letters"]
24 $letters
25 [1] "a" "b" "c"
26
27 > x[["letters"]]
28 [1] "a" "b" "c"
29 > x$letters
30 [1] "a" "b" "c"

```

1.3.8. Фактор

Факторы представляют собой структуру данных для хранения векторов категориальных данных (классов), т.е. величин, которые могут принимать значения из конечного, в общем случае, **неупорядоченного множества**. Факторы создаются с помощью функции *factor(x = character(), levels, labels = levels)*. Предположим, что у нас имеется 3 класса Yes, No, Perhaps. И некоторая выборка из 6 объектов, каждый из которых принадлежит одному из этих классов. Тогда

```

1 > v = c("Yes", "No", "Yes", "Perhaps", "No", "Perhaps")
2 > f = factor(v)
3 > f
4 [1] Yes      No       Yes      Perhaps No       Perhaps
5 Levels: No Perhaps Yes

```

Для получения и изменения текстового вектора, содержащего имена уровней фактора, служит функция *levels(x)*. Несмотря на то, что фактор не является вектором, изменяя с помощью функции *dim(x)* вектор размерностей, **можно превратить фактор в фактор-матрицу и фактор-массив**.

1.3.9. Фрейм (или таблицы данных)

Фреймы данных (*data frames*) один из самых важных типов данных в R, позволяющий объединять данные разных типов вместе. **Фрейм является специальной версией списка, где все элементы имеют одинаковую длину**. Можно считать, что фрейм данных — это двумерная таблица, в которой (в отличие от числовых матриц), разные столбцы могут содержать данные разных типов (но все данные в одном столбце имеют один тип). Например, такая таблица может содержать результаты эксперимента.

Создать фрейм данных можно с помощью функции *data.frame(...)*, аргументами которой являются произвольное количество элементов (столбцов) фрейма. **В качестве элементов фрейма данных могут выступать векторы, факторы, матрицы, списки или другие фреймы**. При этом все векторы

должны иметь одинаковую длину, а матрицы и фреймы одинаковое (такое же) число строк. Функция *data.frame(...)* просто собирает все данные вместе. Символьные векторы конвертируются в факторы. Остальные данные собираются во фрейм такими, какие они есть.

```
1 > a = m a t r i x (1:8 , nrow = 4 , ncol = 2)
2 > b = c ( " a " , " b " , " c " , " a " )
3 > d = (1:4 %% 2 == 0)
4 > e = f a c t o r ( c ( " soft " , " hard " , " soft " , " m e d i u m " ) )
5 > f = data . f r a m e ( a , b , d , e )
6 > f
7   X1 X2 b    d    e
8 1  1  5 a FALSE soft
9 2  2  6 b  TRUE  hard
10 3  3  7 c FALSE soft
11 4  4  8 a  TRUE medium
12 > f[[1]]
13 [1] 1 2 3 4
14 > f[["b"]]
15 [1] a b c a
16 Levels: a b c
17 > f$d
18 [1] FALSE TRUE FALSE TRUE
```

С помощью функции *colnames(x)* можно изменить имена столбцов фрейма, с помощью *rownames(x)* – имена строк.

Фрейм данных идеально подходит для хранения выборки данных, где каждому столбцу соответствует некоторая переменная или признак, а каждой строке отдельный прецедент.

Для загрузки наборов данных из файла в фрейм может быть использована функция *read.table(file, header = FALSE, sep = "", ...)*, которой на вход подается путь к текстовому файлу с данными, в котором значения в каждой строке разделены символом *sep*. Параметр *header* позволяет указать, следует ли интерпретировать первую строку файла, как имена столбцов таблицы.

1.4. Управляющие конструкции

Приведем краткое описание условных конструкций и циклов языка R.

1.4.1. if/else

```
1 if (condition)
2 {
3     # expression
4 }
5
6 if (condition)
7 {
8     # expression
```



```

9 }
10 else
11 {
12     # alternative expression
13 }

```

Условие `condition` в условной конструкции `if`, как и в циклах, может давать в качестве результата логический вектор, однако, в рассмотрение принимается только первая его компонента. Также могут быть полезными функции `any(x)` и `all(x)`, которые, принимая на вход логический вектор, возвращают значение `TRUE` тогда и только тогда, когда хотя бы один или все элементы вектора равны `TRUE` соответственно.

Корректная запись оператора `if`

```

# Если в коде записано:
if (condition == TRUE)
  x <- TRUE else x <- FALSE
# R воспринимает первую строку как закончившуюся.
# Вторая строка начинается с else, который будет истолкован как самостоятельный
оператор.
# Поскольку такого оператора не существует, будет выдано сообщение об ошибке
# Error: unexpected token 'else'
# Чтобы R интерпретировал else как часть предшествующего if'а, нужно указать ему
# при помощи фигурных скобок, что if еще не закончился:
if (condition == TRUE) {
  x <- TRUE
} else {x <- FALSE}

```

1.4.2. `for`

```

1 for (i in I)
2 {
3     print(i)
4 }

```

Переменная *i* последовательно принимает все значения из вектора *I*. Преждевременный выход из любого цикла (не обязательно `for`) осуществляется командой ***break***. Для прерывания текущей итерации и перехода к следующей служит команда ***next***.

1.4.3. `while`

```

1 while (condition)
2 {
3     # expression
4 }

```

1.4.4. `repeat`

```

1 repeat
2 {
3     # expression
4 }

```

Данная конструкция реализует бесконечный цикл, выход из которого можно осуществить с помощью команды `break`.

1.4.5. Функции

Функции в языке *R* определяются с помощью ключевого слова `function`, вслед за которым в круглых скобках идет перечисление формальных аргументов функции. Аргументам можно назначать значения по умолчанию, как выражения, зависящие, в том числе, и от других формальных аргументов. При этом вычисление данного выражения будет производиться только, если значение этого аргумента потребуется при выполнении тела функции. Тело функции идет после указания списка ее формальных аргументов и должно быть заключено в фигурные скобки, если содержит более одного выражения. Значением, возвращаемым функцией, является либо результат последнего выражения в теле функции, либо аргумент функции **`return(x)`**, прерывающей выполнение функции.

Как правило, определенная описанным выше способом функция присваивается некоторому имени с помощью оператора присваивания. Анонимные функции могут быть использованы, как фактические аргументы других функций.

```

1 > sq = function(x)
2 + {
3 +     x * x
4 + }
5 >
6 > fun = function(x, a = x * x)
7 + {
8 +     return(x + a)
9 + }

```

Все аргументы передаются в функцию по значению, а, следовательно, не могут быть изменены в ней. **Для того, чтобы вернуть несколько однотипных значений, следует возвращать вектор, разнотипных – список.**

Вызов функции осуществляется путем указания ее имени с последующим перечислением фактических аргументов в круглых скобках, как это делалось для всех рассмотренных функций. Следует отметить, что помимо указания всех аргументов в порядке, указанном при определении функции, **часть или все аргументы могут быть указаны в именованном виде `argumentName = argumentValue`**, при этом порядок следования не важен.

1.5. Формулы

Зачастую в ходе статистического моделирования необходимо выразить

зависимость одной величины от другой и описать характер данной зависимости. В языке **R** для этого используются специальные объекты, называемые формулами. Для определения формулы используется оператор \sim , синтаксис которого выглядит следующим образом: $y \sim \text{model}$. Слева от оператора \sim указывается имя зависимой переменной, а справа выражение специального вида – модель зависимости. Например, формула $y \sim x_1 + x_2$ определяет линейную зависимость y от x_1 и x_2 : $y = a_0 + a_1x_1 + a_2x_2$. Все переменные, используемые в модели, должны быть либо числовыми векторами, либо факторами. При этом логические векторы преобразуются в факторы.

Рассмотрим правила формирования модели формулы. Знаки $+$ и $-$ определяют вхождение или исключение определенной переменной как линейного члена в модель, т.е. в данном случае семантика данных операций отлична от просто арифметического сложения и вычитания значений переменных. При необходимости арифметические операции могут быть использованы в модели с помощью функции $I(x)$, которая позволяет интерпретировать свой аргумент, как обычное выражение языка **R**. Также преобразования (не только арифметические) переменных могут быть предварительно заданы отдельными функциями, которые можно впоследствии использовать в модели формулы, например, $\log(x)$ или $f(x_1, x_2)$.

Оператор: при использовании в модели формулы обозначает взаимное влияние нескольких переменных. Например, если x_1 и x_2 числовые векторы, то член $x_1:x_2$ равносителен $I(x_1 * x_2)$. В случае, если x_1 или/и x_2 является фактором, то отдельно рассматриваются все его уровни. Например, если x_1 – фактор с двумя уровнями a и b , а x_2 – числовой вектор, то x_1 заменяется на два числовых 0-1 вектора

$z_1 = \text{as.numeric}(x_1 == "a")$ и $z_2 = \text{as.numeric}(x_1 == "b")$

и модель эквивалентна $z_1:x_2 + z_2:x_2$.

Оператор $*$ обозначает взаимодействие и главные эффекты, т.е. влияние самих переменных и их совместное влияние.

$x_1 * x_2$ эквивалентно $x_1 + x_2 + x_1:x_2$,

а $x_1 * x_2 * x_3$ эквивалентно $x_1 + x_2 + x_3 + x_1:x_2 + x_1:x_3 + x_2:x_3 + x_1:x_2:x_3$.

Оператор $^$ обозначает главные эффекты и все взаимодействия вплоть до указанного порядка. $(x_1 + x_2 + x_3)^2$ эквивалентно $x_1 + x_2 + x_3 + x_1:x_2 + x_1:x_3 + x_2:x_3$.

1.6. Пакеты расширений

Одной из причин популярности языка **R** является обилие бесплатных пакетов (packages), расширяющих базовые возможности языка. Основным репозиторием пакетов расширений является репозиторий CRAN (<http://cran.r-project.org>), на настоящий момент насчитывающий более 4000 пакетов.

Для использования определенного пакета, требуется установить его со всеми зависимостями и подключить. Для того, чтобы установить пакет, можно скачать его из репозитория в бинарном виде или в виде исходного кода (в этом случае

придется выполнять сборку пакета вручную), а затем установить с помощью функции `install.packages(pkgs, lib)`, первым аргументом которой является путь к zip-файлу с пакетом в бинарном виде, а вторым – путь к директории (библиотеке), в которую будет установлен данный пакет. Список известных библиотек можно посмотреть, вызвав функцию `.libPaths()`.

Если при вызове `install.packages()` параметр `lib` не был указан, то используется первая компонента вектора `.libPaths()`. Другим способом установки пакета является его автоматическая загрузка из репозитория, для этого в качестве параметра `pkgs` функции `install.packages(pkgs)` требуется указать только имя пакета, который в случае удачного поиска в репозитории будет скачан и установлен вместе со всеми зависимостями.

Посмотреть список установленных пакетов можно с помощью функции `library()`. Чтобы загрузить пакет для использования, необходимо вызвать функцию `library(package)` или `require(package)`, где `package` – имя пакета. Чтобы посмотреть краткую справку об установленном пакете можно вызвать функцию `library(help = package)`.

Обновить установленные пакеты можно функцией `update.packages()`.

1.7. Графика

Для изображения графиков в **R** есть функция `plot(x, y, ...)`. Здесь `x` – вектор значений абсцисс и `y` – вектор значений ординат.

Функция `plot` допускает множество дополнительных параметров. Приведем описание некоторых из них:

- `log = "x"` или `log = "y"` или `log = "xy"` делает соответственно масштаб по оси абсцисс, ординат или по обеим осям логарифмическим;
- `type` позволяет указать тип выводимого графика. Некоторые значения параметра: `"p"` (точки, по умолчанию), `"l"` (линии), `"b"` (линии и точки), `"o"` (линии и точки перекрываются), `"n"` (ничего не рисуется);
- `xlab` и `ylab` позволяет указать подписи к оси абсцисс и ординат соответственно;
- `main` и `sub` создают надписи сверху и снизу графика;
- `col` задает цвет графика. Некоторые возможные значения: `"blue"`, `"red"`, `"green"`, `"cyan"`; `"magenta"`, `"yellow"`, `"black"`. Можно задать цвет rgb- вектором функцией `rgb(r, g, b)`, где `r`, `g` и `b` от 0 до 1;
- `lty` определяет стиль линии. Некоторые возможные значения `"blank"` (нет линии), `"solid"` (сплошная линия), `"dashed"` (пунктирная линия), `"dotted"` (точечная линия), `"dotdash"` (точка-тире).

Следующие функции никогда не стирают имеющегося изображения. Им могут быть переданы также дополнительные параметры, аналогичные соответствующим параметрам функции `plot`.

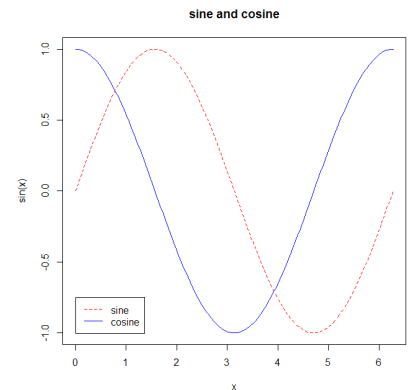
- `points(x, y, ...)` рисует дополнительные точки, `lines(x, y, ...)` рисует ломаную линию, соединяющую указанные точки.

- `text(x, y = NULL, labels = seq_along(x), ...)` добавляет текст. По умолчанию он центрируется вокруг точки (x, y). Указанные параметры могут быть векторами. В этом случае будет размещено несколько надписей.

- `abline(k, b, ...)` рисует прямую $y = kx + b$.

- `abline(h = y, ...)` и `abline(v = x, ...)` отрисовывают горизонтальные и вертикальные прямые соответственно.

- `legend(x, y = NULL, legend, ...)` добавляет легенду в указанную точку. Рассмотрим пример:



```
1 > x = seq(0, 2 * pi, length = 250)
```

```
2 > plot(x, sin(x), col = "red", type = "l", main = "sine and cosine", lty = "dashed")
```

```
3 > lines(x, cos(x), col = "blue")
```

```
4 > legend(0, -0.5, legend = c("sine", "cosine"), lty = c("dashed", "solid"), col = c("red", "blue"))
```

Результат выполнения данного набора команд приведен на рис. справа

1.8. Задания к лабораторной работе общее для всех

Внимание: циклы не использовать!

1. Создайте вектор p , состоящий из значений 7, 6, 5, 4, и вектор q , состоящий из 0, 1, 2, 3, произведите сложение, вычитание, умножение, деление, возведение в степень этих векторов.

2. Создайте вектор 0, 2, 0, 4, ..., 18, 0, 20 наиболее коротким способом. Создайте вектор, в котором будут содержаться первые 20 степеней двойки. Создайте вектор из чисел 1, 10, 100, 1000, 10000.

3. С помощью функции `sum()` посчитать сумму последовательностей:

$$\frac{1}{1 \cdot 2} + \frac{1}{2 \cdot 3} + \frac{1}{3 \cdot 4} + \dots + \frac{1}{50 \cdot 51}.$$

$$\frac{1}{2^0} + \frac{1}{2^1} + \frac{1}{2^2} + \frac{1}{2^3} + \dots + \frac{1}{2^{20}}.$$

$$1 + \frac{4}{3} + \frac{7}{9} + \frac{10}{27} + \frac{13}{81} + \dots + \frac{28}{19683}$$

Выведите число значений в крайней последовательности больше 0,5.

4. Создайте вектор `vec3` со значениями 3, 6, 9, ..., 24, 27 с помощью функции `seq()` с нужными параметрами. Извлеките 2, 5 и 7 значения вектора, извлеките предпоследнее значение вектора (использовать функцию `length()`), а затем все, кроме предпоследнего. Верните все элементы вектора, кроме шестого. Попробуйте извлечь сотое значение вектора. Верните все значения вектора `vec3` кроме первого и последнего. Найдите все значения вектора `vec3`, которые больше

4, но меньше 10. Найдите все значения вектора `vec3`, которые меньше 4 или больше 10.

1.9 Задания к лабораторной работе по варианту

Выберите себе 2 задания из списка по принципу:

1) №варианта%15

2) №варианта%15+15 и решите их ^{*)}

***) Вариант может быть изменен по согласованию с преподавателем.**

1. Дан вектор `g`, в котором хранятся следующие значения:

1, 0, 2, 3, 6, 8, 12, 15, 0, NA, NA, 9, 4, 16, 2, 0

Создайте вектор `g`. Используя `R`, выведите на экран:

1. первый элемент вектора;
2. последний элемент вектора;
3. элементы вектора с третьего по пятый включительно;
4. элементы вектора, которые равны 2;
5. элементы вектора, которые больше 4;
6. элементы вектора, которые кратны 3 (делятся на 3 без остатка);
7. элементы вектора, которые больше 4 и кратны 3;
8. элементы вектора, которые или меньше 1, или больше 5;
9. индексы элементов, равных 0;
10. индексы элементов, которые не меньше 2 и не больше 8.

2. Есть набор данных:

```
>df <- data.frame(var1=c(11,21,31), var2=c(12,22,32),  
var3=c(13,23,33), var4=c(14,24,34), row.names=c("case1",  
"case2", "case3"))
```

```
>df
```

	var1	var2	var3	var4
case1	11	12	13	14
case2	21	22	23	24
case3	31	32	33	34

- 1.1. Выбрать значения `var1`, `var2`, `var3` для `case1`;
- 1.2. Выбрать значения всех переменных для `case2`, которые больше 22;
- 1.3. Выбрать имена переменных для колонок 1 и 3;
- 1.4. Добавить колонку с именем `Y` и значениями -1, 0, 1;
- 1.5. Удалить строку `case2`;
- 1.6. Значения второй колонки возвести в третью степень.

3. Создайте фрейм данных из $N = 20$ записей со следующими полями: *Nrow*

– номер записи, *Name* – имя сотрудника, *BirthYear* – год рождения, *EmployYear* – год приема на работу, *Salary* – зарплата. Заполните данный фрейм данными так, что *Nrow* изменяется от 1 до N, *Name* задается произвольно, *BithYear* распределен равномерно (случайно) на отрезке [1960, 1985], *EmployYear* распределен равномерно на отрезке [*BirthYear* + 18, 2006], *Salary* для работников младше 1975 г.р. определяется по формуле $Salary = (\ln(2007 - EmployYear) + 1) * 8000$, для остальных $Salary = (\log_2(2007 - EmployYear) + 1) * 8000$.

Подсчитайте число сотрудников с зарплатой, большей 15000. Добавьте в таблицу поле, соответствующее суммарному подоходному налогу (ставка 13%), выплаченному сотрудником за время работы в организации, если его зарплата за каждый год начислялась согласно формулам для *Salary*, где вместо 2007 следует последовательно подставить каждый год работы сотрудника в организации.

4. Есть набор данных:

```
>df <- data.frame(var1=c(1,2,3), var2=c(4,5,6),
var3=c(7,8,0), var4=c(9,10,11), row.names=c("case1",
"case2", "case3"))
```

```
>df
```

```
      var1 var2 var3 var4
case1    1   4   7   9
case2    2   5   8  10
case3    3   6   0  11
```

- 1.1. Выбрать значения var1, var2, var3 для case1;
- 1.2. Выбрать значения всех переменных для case3, которые меньше 8;
- 1.3. Выбрать имена переменных для колонок 2 и 4;
- 1.4. Добавить колонку с именем Y и значениями -10, 0, 11;
- 1.5. Удалить строку case2;
- 1.6. Значения второй колонки возвести в третью степень.

5. Напишите код, который заменяет случайно элементы некоторого вектора на пропущенные значения (NA). Ваш код должен работать для вектора любой длины. Далее необходимо вывести на экран индексы пропущенных значений в векторе. Напишите код, который считает, сколько пропущенных значений в векторе.

6. Напишите код, который позволяет создать вектор из id (уникальных номеров) респондентов, если известно, что в опросе участвовало 100 респондентов. То же задание, но известно лишь, что число респондентов <=200 и это число вводится пользователем.

7. Решение задачи предполагает рациональное решение с помощью функций R, не нужно создавать вектора перечислением большого числа элементов вручную.

	country	year
1	France	2000
2	France	2001
3	France	2002
4	France	2003
5	France	2004
6	Italy	2000
7	Italy	2001
8	Italy	2002
9	Italy	2003
10	Italy	2004
11	Spain	2000
12	Spain	2001
13	Spain	2002
14	Spain	2003
15	Spain	2004

Известно, что в таблице хранятся показатели по 3 странам за 5 лет (см. рис).

Создайте вектор, который мог бы послужить столбцом `country` в этой таблице. Создайте вектор, который мог бы послужить столбцом `year` в этой таблице.

Таблица выглядит так:

8. Исследователь решил создать вектор `income`, в котором сохранены доходы нескольких респондентов:

```
income <- c(10000, 32000, 28000, 150000, 65000, 1573)
```

Исследователю нужно получить вектор `income_class`, состоящий из 0 и 1: 0 ставится, если доход респондента ниже среднего дохода, а 1 — если больше или равен среднему доходу. Создайте вектор `income_class`.

Подсказка: сначала можно посчитать среднее значение по вектору `income` и сохранить его в какую-нибудь переменную. Можно воспользоваться встроенной функцией `mean()`.

9. Дан вектор случайных чисел `v`, длиной 100. Найти в `v` те элементы, значение которых меньше среднего арифметического, взятого от всех элементов вектора `v`. Сформировать из `v` матрицу 10x10 и вывести на экран векторы четных и нечетных строк.

10. Создать случайный вектор `vest` натуральных чисел от 1 до 100 определить, сколько из них кратны любому из чисел в диапазоне от 2 до 5. Числа для определения кратности вводить с клавиатуры.

11. Дан вектор случайных чисел `v`, длиной 100. В векторе `v` найти сумму элементов, находящихся между минимальным и максимальным элементами. Сами минимальный и максимальный элементы в сумму не включать, циклы не использовать.

12. В векторе случайных целых чисел в диапазоне `[-20; 50]` поменять местами минимальный и максимальный элементы. Циклы не использовать. Подсчитать количество отрицательных чисел, заменить их нулями.

13. В диапазоне натуральных чисел от 2 до 99 определить, сколько из них кратны любому из чисел в диапазоне от 2 до 9.

14. Создать вектор из 20 элементов в диапазоне значений от -15 до 14 включительно. Определить количество элементов по модулю больших, чем максимальный. Используйте векторные вычисления R и не используйте циклы. Определите есть ли в векторе числа 3, 5, 7 и сколько раз они встречаются.

15. Ввести данные с клавиатуры, сформировать `DataFrame`. Найти планету самую удаленную и самую близкую к Солнцу. Упорядочить `DataFrame` по возрастанию расстояния от Солнца. Посчитать среднюю относительную массу

планет. Вывести название столбцов.

Планета	Расст. до Солнца	Относ. объем	Относ. масса
Меркурий	57.9	0.06	0.05
Венера	108.2	0.92	0.81
Земля	149.6	1.00	1.00
Марс	227.9	0.15	0.11
Юпитер	978.3	1345.00	318.40
Сатурн	1429.3	767.00	95.20

Вторые задания

16. Создать матрицу `matr [m,n]`. Значения `m` и `n` ввести с клавиатуры. Присвоить имена строкам и столбцам. Сформировать субматрицу из четных строк. Найти вектор максимальных элементов столбцов матрицы.

17. Дан вектор случайных чисел `vec`, длиной 80. Найти в `v` те элементы, значение которых больше среднего арифметического, взятого от всех элементов вектора `v`. Сформировать из `v` матрицу `8x10` и вывести на экран векторы четных и нечетных столбцов.

18. Создать `n` векторов с разными типами данных, (`n` ввести с клавиатуры), определить класс каждого вектора, объединить их в `data.frame`, присвоить имена столбцам, определить число строк и столбцов созданного набора данных.

19. Сформируйте набор данных, который содержит цвет ветки метро и соответствующую длину этой ветки (не менее 7 веток метро). Необходимо выдать название самой длинной и самой короткой веток метро. С помощью функции `paste()` сформируйте ответ вида «Синяя ветка Московского метро — самая длинная. Ее протяженность составляет 45 км».

20. Сформируйте набор данных, который содержит название фрукта и соответствующий ему вес в граммах (не менее 8 фруктов). Из `data.frame` необходимо сформировать таблицу фруктов с одинаковыми весами. Определить число строк и столбцов созданного набора данных.

21. Создать 5 векторов с характеристиками автомобилей — вес, емкость двигателя, тип коробки передач, максимальная скорость, тип кузова. Создать из них `data.frame`, в качестве названий присвоить марки автомобилей. Выбрать только автомобили с одинаковым объемом двигателя, упорядочить их по алфавиту. Определить число строк и столбцов созданного набора данных.

22. Создать произвольную матрицу `10x10`. Найти в ней позиции нулей, вывести матрицу индексов нулей. Транспонировать матрицу, вывести новую матрицу, содержащую только четные строки транспонированной.

23. Создайте `data.frame temp`, в котором хранятся значения среднемесячных температур воздуха в городе Краснодаре. Напишите код, который вычисляет следующие вектора:

- *количественное изменение* температуры от месяца к месяцу (в градусах)
- номера и названия *зимних* месяцев (со среднемесячной температурой ниже нуля);

Максимальная и минимальная среднемесячная температура ^[1]												
Месяц	Янв	Фев	Мар	Апр	Май	Июн	Июл	Авг	Сен	Окт	Ноя	Дек
Самый тёплый, °C	7,5 (1915)	7,1 (2016)	10,0 (2008)	16,5 (2012)	21,8 (2013)	25,3 (2019)	27,5 (2001)	27,7 (2006, 2010)	23,2 (2015)	16,8 (2012)	12,0 (2010)	7,2 (2010)
Самый холодный, °C	-10,8 (1972)	-13,1 (1954)	-3,7 (1929)	7,2 (1929)	13,3 (1919)	17,6 (1933)	20,6 (1956)	19,4 (1984)	13,9 (1884)	6,4 (1951)	-4,0 (1993)	-7,0 (1920)

24. На местности задан прямоугольник с координатами левого нижнего (x_1, y_1) и правого верхнего (x_2, y_2) угла. Напишите программу, которая размещает внутри этого прямоугольника случайным образом N точек и представляет результат в виде матрицы координат `coords` с двумя столбцами и N строками. Вызовите в конце программы `plot(coords)`, чтобы посмотреть на результат. Исходные координаты вводить с клавиатуры.

Подсказка: координаты случайно размещенных точек имеют равномерное распределение. Вам необходимо сначала сформировать случайные векторы координат X и Y , и после этого объединить их в матрицу.

25. Высотная поясность на северном склоне Западного Кавказа, согласно Большой Российской энциклопедии устроена следующим образом:

- до 500 м — степь и лесостепь
- до 800 м — низкогорные широколиственные леса (дуб, граб)
- до 1300 м — среднегорные широколиственные леса (бук)
- до 1600 м — смешанные леса (ель, пихта, бук)
- до 2300 м — криволесья (береза, бук, клён)
- до 2500 м — субальпийские и альпийские луга
- до 3300 м — субнивальная зона (фрагментарная растительность)
- выше (условно до 5000 м) — гляциально-нивальная зона

Создайте фрейм данных, включающий три столбца: минимальная высота пояса (H_{min}), максимальная высота пояса (H_{max}) и название высотного пояса ($Zone$ – можно закодировать для краткости). Минимальную высоту надо вычислить на основе максимальной, приняв, что для нижнего пояса она условно равна 400 м.

Напишите программу, которая просит пользователя ввести высоту и возвращает высотный пояс, соответствующий введенной высоте (достаточно вывести строчку фрейма данных).

26. Создать 5 векторов с характеристиками мобильных телефонов – вес, емкость аккумулятора, диагональ, поддерживаемый стандарт Wi-Fi, тип OS. Создать из них data.frame, в качестве названий присвоить модели телефонов. Ввести с клавиатуры требуемый размер диагонали и найти в data.frame соответствующую модель и ее позицию в df. Определить число строк и столбцов созданного набора данных. Выбрать только Android – устройства.

27. Создать df, содержащий себестоимость добычи нефти по странам мира в 2021 году. Отдельно создайте столбец с указанием континента и добавьте его к готовому df. Отсортируйте его по возрастанию стоимости. Выберите в новый df только страны Европы, в другую выборку – страны Америки.

Страны мира	Себестоимость 1 барреля добытой нефти, долл. США
США (сланцевая нефть)	20
США (шельф Мексиканского моря)	25
Норвегия (Северное море)	17
Канада (битумная нефть)	16
Россия (новые месторождения)	16
Нигерия	11
Мексика	9
Венесуэла (битумная нефть)	9
Алжир	8
Ливия	7
Россия (действующие проекты)	6
Казахстан	6
Иран	5
Саудовская Аравия	4

28. Создать df, содержащий рейтинг стран по индексу и расходам на образование в 2020г. Отдельно создайте столбец с указанием континента и добавьте его к готовому df. Пропущенные данные заполните средним значением по столбцу. Отсортируйте его по возрастанию расходов на ВВП. Выберите в новый df только страны Европы, в другую выборку – страны Америки.

Ранг	Страна	Индекс образования	Расходы, % ВВП
1	Германия	0.946	4.6
2	Австралия	0.923	5.1
3	Новая Зеландия	0.923	7.2
4	Дания	0.92	8.7
5	Норвегия	0.919	7.3
6	Исландия	0.918	7.8
7	США	0.899	5.4
8	Канада	0.891	4.8
28	Япония	0.85	3.8