Федеральное государственное образовательное бюджетное учреждение
высшего профессионального образования
«ФИНАНСОВЫЙ УНИВЕРСИТЕТ
ПРИ ПРАВИТЕЛЬСТВЕ РОССИЙСКОЙ ФЕДЕРАЦИИ»
(Финансовый университет)

Департамент анализа данных, принятия решений
и финансовых технологий

Дисциплина «Программирование в среде R»

П.Б. Лукьянов

МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ
ЛАБОРАТОРНОЙ РАБОТЫ № 5

Основные управляющие конструкции языка R

Для студентов, обучающихся по направлению подготовки
«Прикладная информатика»
(программа подготовки бакалавра)

Москва 2020

Цель лабораторной работы – изучение основных управляющих конструкций языка R и использование этих конструкций для реализации логики решения практических задач в прикладных программах.

В любом языке программирования присутствует несколько базовых языковых конструкций, управляющих логикой работы программы. Эти управляющие конструкции дают возможность программисту реализовать на практике вычислительный алгоритм любой сложности. В данной лабораторной работе рассматриваются конструкции условного выбора и реализация циклов.

Конструкция условного выбора if – else может быть реализована в двух формах, неполной (рис. 1) и полной (рис. 2):

```
a<- 5
b<- 10
if (a<b) {
   print("a меньше b")
}
```

Рис. 1. Неполная форма ветвления

```
a<- 5
b<- 10
if (a<b) {
   print("a меньше b")
} else {
   print("a больше или равно b")
}
```

Рис. 2. Полная форма ветвления

Формальное описание конструкции условного выбора следующее:

- после ключевого слова if в круглых скобках пишется логическое выражение, результат которого равен либо TRUE, либо FALSE

- сразу за логическим выражением в фигурных скобках следует код, который будет выполняться, если результат вычисления логического выражения равен TRUE
- если результат логического выражения равен FALSE, то выполняется код, следующий за ключевым словом else. Этот код также заключается в фигурные скобки (рис. 3).

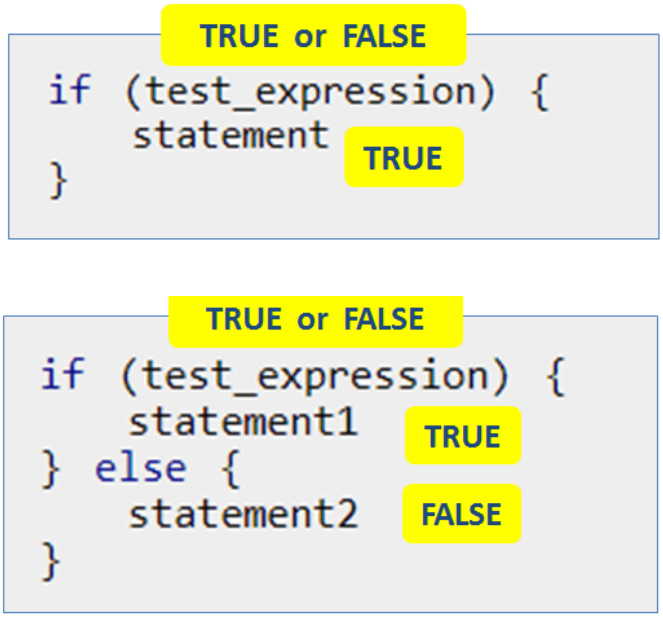

Рис. 3. Логика выполнения программы при ветвлении

Возможны вложенные конструкции if – else (рис. 4, 5):

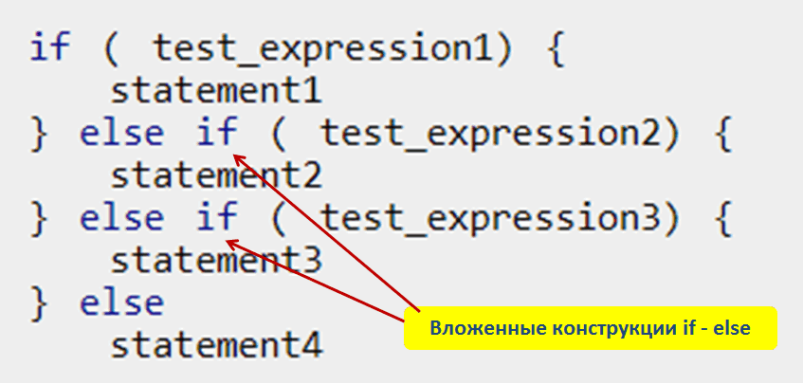

Рис. 4. Программная реализация проверки нескольких условий

```r
x <- 0
if (x < 0) {
    print("Negative number")
} else if (x > 0) {
    print("Positive number")
} else
    print("Zero")
```

Рис. 5. Пример проверки нескольких условий

При использовании в условии логических выражений с векторами (см. пример рис. 6) учитывается результат выражения только **для первых элементов векторов** (рис. 7), причем сначала вектора приводятся к одной размерности по следующему правилу: элементы меньшего вектора последовательно дублируются до тех пор, пока размерности двух векторов не сравняются.

```r
a <- c(0:15:0, FALSE)
if (a + 2 & (-1:3)) {
  print("Выполняется код на условие TRUE")
}
else {
  print("Выполняется код на условие FALSE")
}
```

Рис. 6. Логическое условие с векторами разной длины

Проверка условий по всем элементам вектора, функция ifelse()

Для работы **со всеми элементами вектора** при проверке условий предназначена функция **ifelse()**, формальное описание которой и простейший пример представлены на рис. 8. Пример с векторами разной длины показан на рис. 9. Результатом работы функции будет вектор

(-10, -9, 5, -7, -6)

```
> a <- c(0:15:0, FALSE)
Warning message:
In 0:15:0 : numerical expression has 16 elements: only the first used
> if (a + 2 & (-1:3)) {
+   print("Выполняется код на условие TRUE")
+ } else {
+   print("Выполняется код на условие FALSE")
+ }
[1] "Выполняется код на условие TRUE"
Warning messages:
1: In a + 2 & (-1:3) :
  longer object length is not a multiple of shorter object length
2: In if (a + 2 & (-1:3)) { :
  the condition has length > 1 and only the first element will be used
>
```

Рис. 7. Вывод результата и предупреждения среды R

ifelse(test_expression, x, y)

test_expression – логический вектор или выражение, приводимое к логическому вектору
Возвращаемые значения **x, y** – вектора той же длины, что и **test_expression**

Если (**test_expression[i] == TRUE**), возвращается **x[i]**, иначе **y[i]**.

Пример ifelse()

```
> a = c(5,7,2,9)
> ifelse(a %% 2 == 0,"even","odd")
[1] "odd"  "odd"  "even" "odd"
```

Рис. 8. Описание и пример работы функции ifelse()

```
a <- c(5,7,2,9, TRUE)
b <- ifelse (a%%2 == 0, (3:7), (-10:10))
b
```

Рис. 9. Пример ifelse() с векторами разной длины

Циклы

Циклы дают возможность выполнять определенные строки кода необходимое число раз. Выход из цикла происходит по наступлению некоторого условия. При другом варианте цикла (цикл for()) выполняется последовательный обход элементов вектора. В этом случае цикл завершается при достижении последнего элемента.

Алгоритмические особенности вариантов циклов привели к необходимости использования различных конструкций и разных ключевых слов для управления циклами. Вместе с тем, суть циклов описывается в терминах, представленных на рис. 10.

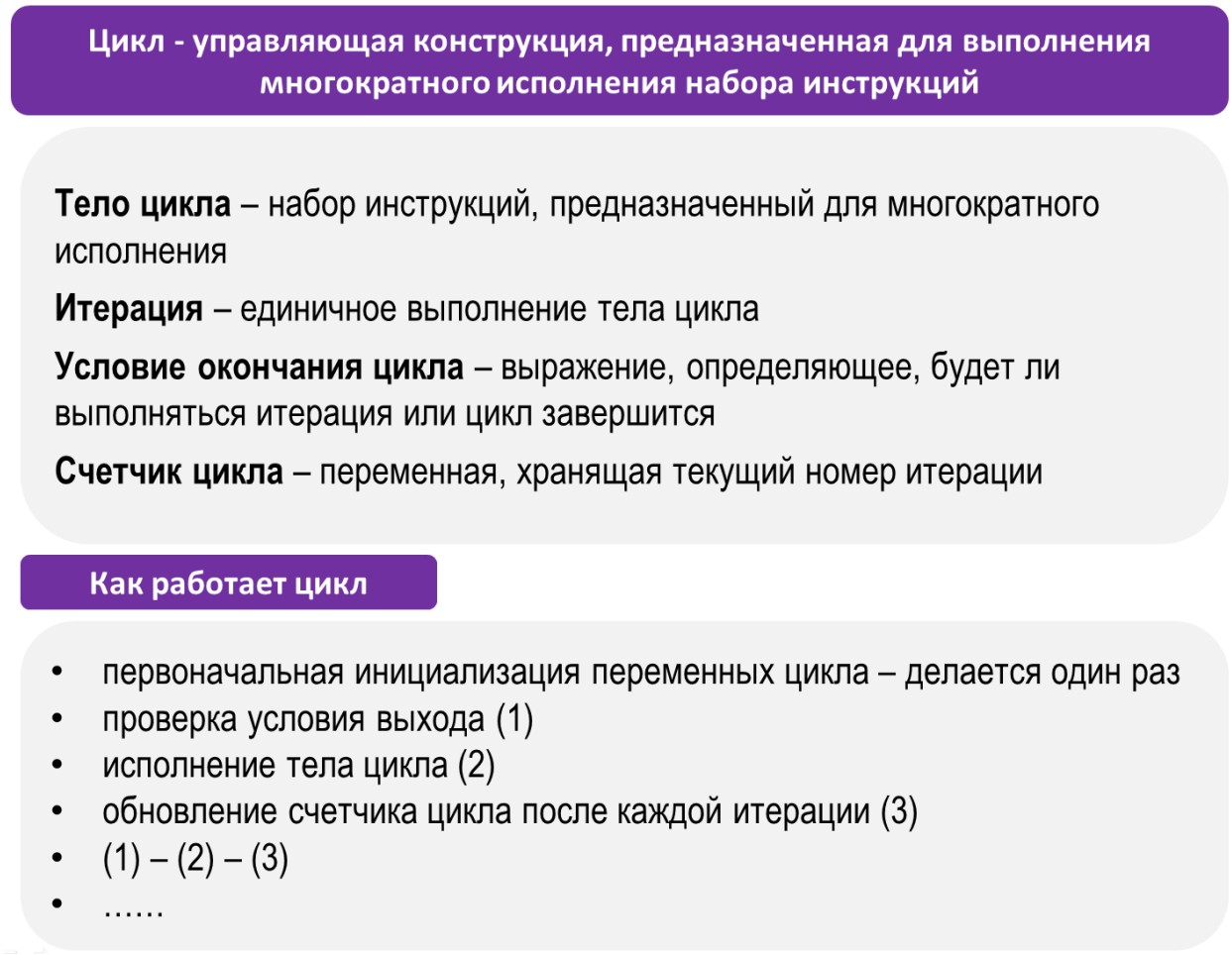

Рис. 10. Определение терминов и описание работы цикла

Цикл **for()**

Логика работы цикла с использованием конструкции for() отражена на рис. 11.

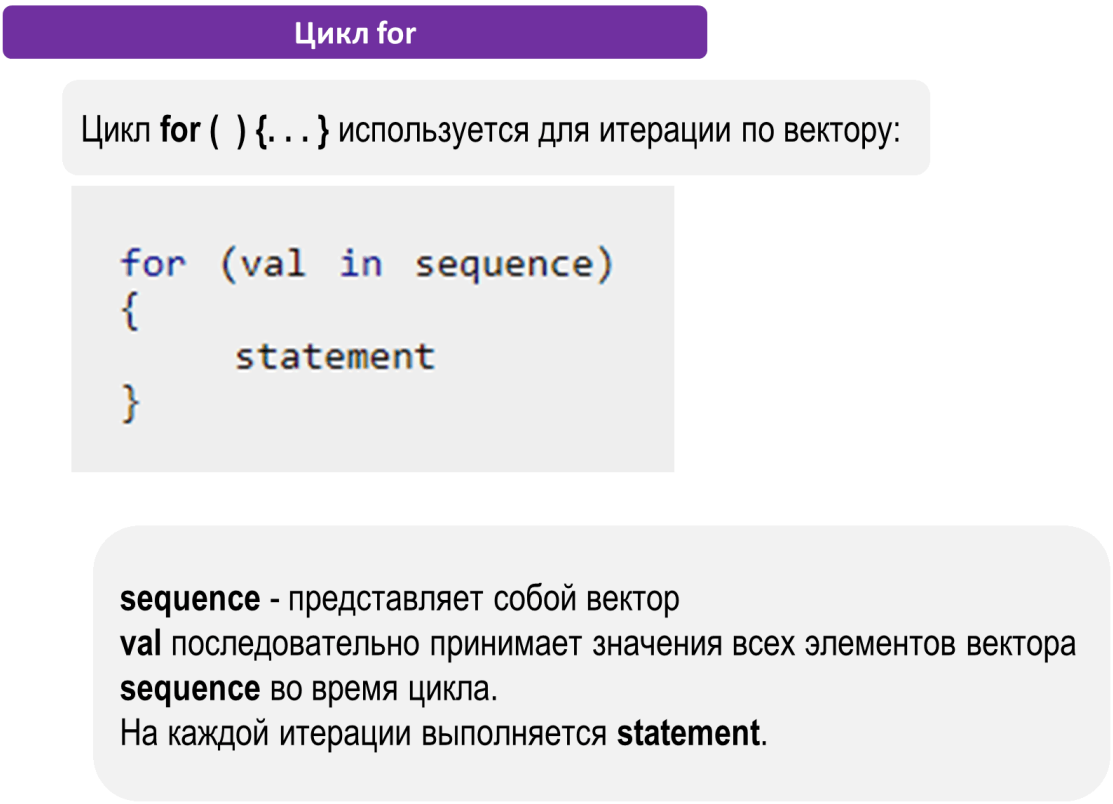

Рис. 11. Описание работы цикла for()

Цикл **while()**

Цикл с предусловием while() работает до тех пор, пока истинно логическое условие, проверяемое перед выполнением каждой итерации. Описание работы цикла приведено на рис. 12.

Принудительный выход из цикла: оператор **break**

Если требуется выйти из цикла при наступлении некоторого условия, вызывается оператор break (прерывание), и начинают выполняться операторы, следующие за телом цикла (рис. 13).

while() – цикл с предусловием

Цикл с предусловием — цикл, который выполняется, пока истинно некоторое условие, указанное перед его началом.

Это условие проверяется до выполнения тела цикла, поэтому тело может быть не выполнено ни разу, если предусловие ложно.

В большинстве языков программирования реализуется оператором while, отсюда его второе название — while-цикл.

```
while (test_expression)
{
    statement
}
```

test_expression – логическое выражение, оценивается перед началом цикла
Если **(test_expression == TRUE)** выполняется первая итерация, выполняется **statement**.
Снова оценивается **test_expression,** если **(test_expression == TRUE)**, **statement** выполняется еще раз. И т.д.

Рис. 12. Описание работы цикла while()

Из цикла можно выйти по условию

Оператор **break** может использоваться внутри цикла (**repeat, for, while**), чтобы остановить итерации и передать управление операторам, выполняющимся после цикла.

В ситуации с вложенными циклами, где есть цикл внутри другого цикла, по оператору **break** выполняется выход из внутреннего цикла во внешний.

```
x <- 1:5
for (val in x) {
     if (val == 3){
        break
    }
   print(val)
}
```

Рис. 13. Цикл выполнится 3 раза

Принудительный переход на следующую итерацию: оператор **next**

Для перехода на следующую итерацию цикла при наступлении некоторого условия используется оператор **next** (рис. 14).

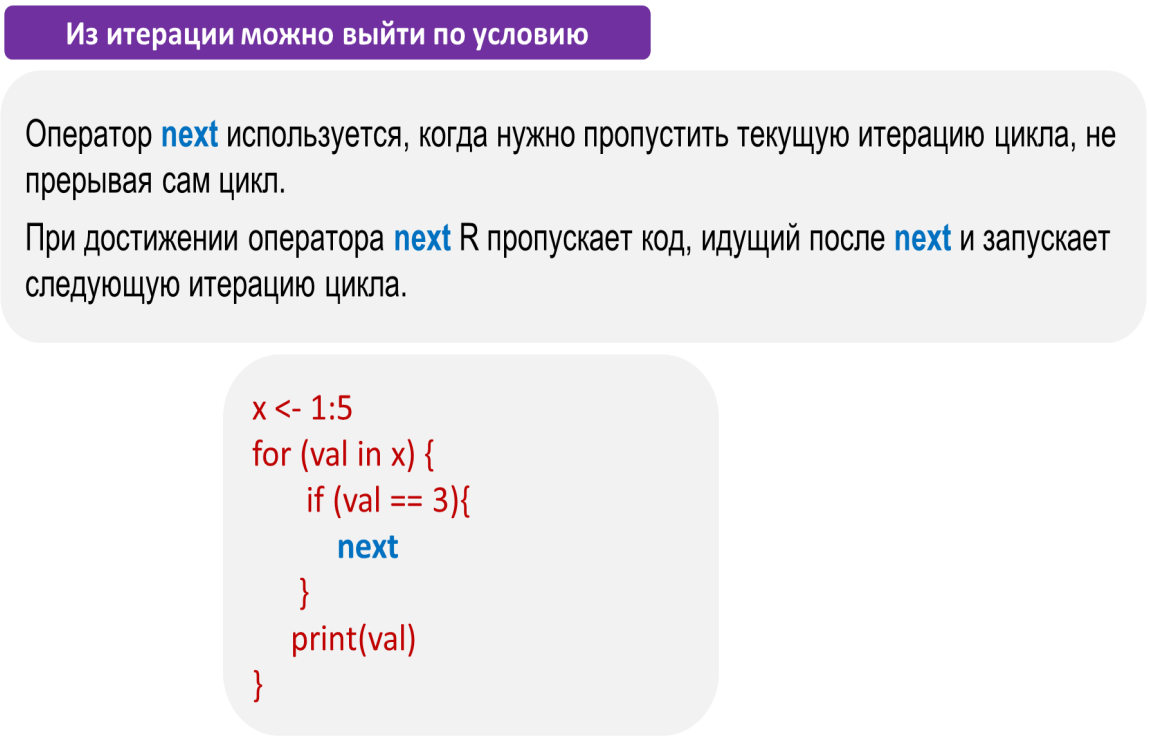

Рис. 14. Цикл выполнится 5 раз, печать выполнится 4 раза

Бесконечный цикл **repeat()**

В языке R есть конструкция для реализации бесконечного цикла. Для того, чтобы выйти из этого цикла, на одной из итераций необходимо вызвать break, иначе программа будет работать вечно. Этот цикл создается с помощью ключевого слова **repeat** (см. рис. 15).

Вопрос выбора той или иной конструкции для реализации цикла не так важен, поскольку в программировании один и тот же результат может быть получен несколькими способами. Обычно после того, как результат достигнут, и программа работает правильно, появляются идеи по упрощению решения, по написанию более изящного и лаконичного кода.

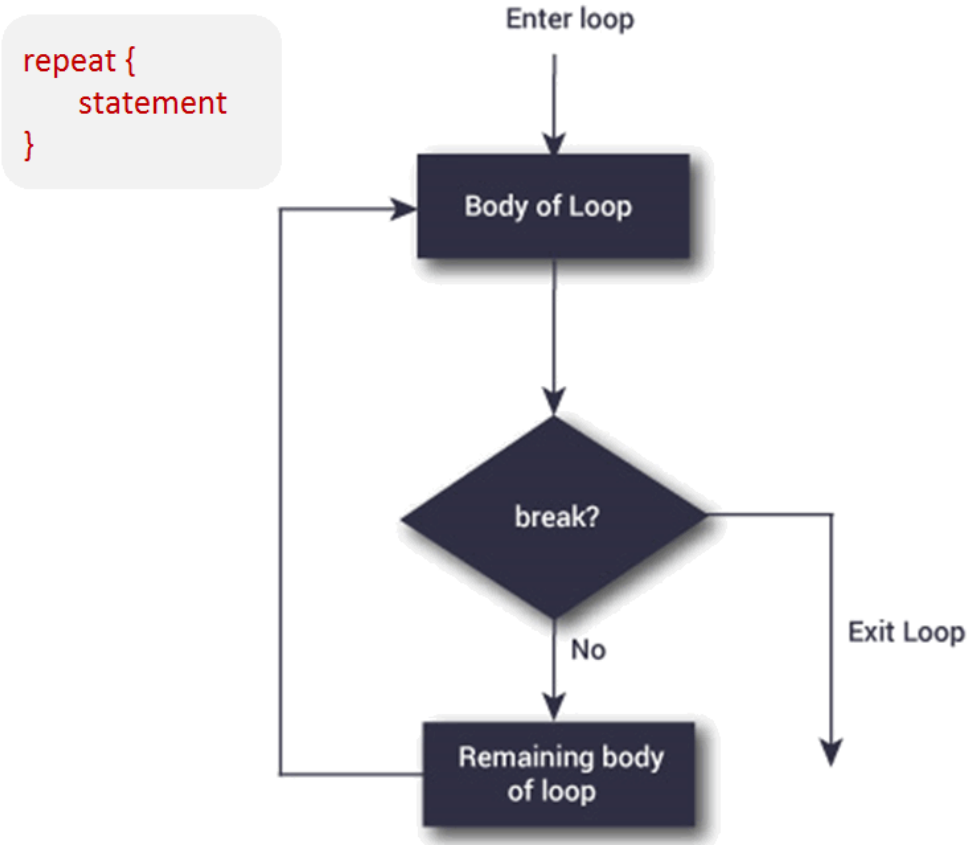

Рис. 15. Бесконечный цикл repeat

Самостоятельные задания

Во всех заданиях, кроме 5, проверку ввода числа не делать

Задание 1.

С клавиатуры ввести целое число N. Рассчитать соответствующий числу день недели, месяц и год, исходя из того, что в каждом месяце 30 дней, 1 задает понедельник января, 2 –вторник января и т.д. Вывести результат в виде строки следующего шаблона:

«Введенное значение N соответствует среде марта, 22-й год».

Задание 2.

Используя цикл, с клавиатуры ввести 8 чисел. Расположить их по убыванию. Результат вывести в виде строки типа «27 > 12 > 5 > -2 …»

Задание 3.

Требовать от Пользователя ввод чисел до тех пор, пока он не наберет слово «Стоп» в любом регистре, в любом сочетании заглавных и прописных букв. Использовать функцию toupper() или tolower().

Задание 4.

Ввести с клавиатуры число N. Подсчитать сумму 1 + 2 + … + N. Вывести на экран введенное число и результат расчета.

Задание 5.

Ввести с клавиатуры число N. Проверить, является ли N целым числом.

Задание 6.

Ввести с клавиатуры два числа. Вывести все числа в диапазоне от большего к меньшему, которые делятся на 3.

Задание 7.

Решить задание № 1 с учетом фактических дней в месяцах: январь – 31, февраль – 28 и т.д.