

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 № 3

Скрипты и первые программы на R

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

Москва 2020

Технология консольного ввода команд, рассмотренная ранее, хороша для выполнения простейших вычислений. Для выполнения сложных расчетов и профессионального программирования нужны более удобные способы и технологии, и именно о них сейчас пойдет речь.

Цель лабораторной работы заключается в изучении пакетной обработки команд с использованием скриптов R и в написании простейших программ.

1. Технология пакетной обработки исходного кода

Любая достаточно сложная вычислительная задача требует для своего решения целой серии вычислений, преобразований, логических операций. Для выполнения таких расчетных работ требуются алгоритмы, реализованные в виде подробного пошагового сценария, или скрипта (от англ. script - сценарий).

Технология работы со скриптами выглядит следующим образом:

- С помощью специализированного редактора в обычный текстовый файл последовательно записываются все необходимые действия для решения задачи (инструкции в виде операторов языка). Команды, как правило, размещаются в файле построчно. В этом же файле пишутся комментарии.
- Затем файл сохраняется на компьютере, ему присваивается имя и задается тип (расширение, суффикс). Именно по расширению файла операционная система понимает, как надо обрабатывать файл, какую программу использовать для его обработки. В случае необходимости файл может неоднократно редактироваться и дополняться.
- Для запуска программы файл загружается в специальную программную среду, проверяется в ней, и затем с помощью специальных команд код, который записан в этом файле, исполняется.

- Данные для обработки хранятся в файлах определенного типа, определенной структуры (т.н. базы данных). Программный код обращается к базе данных для считывания содержимого и копирования данных в переменные программы.
- Для случаев сложных вычислений, для больших программ разные куски кода могут записываться в несколько файлов. С помощью определенных конструкций языка последовательность вычислений или выполнение программы будут происходить по определенному сценарию, с последовательным обращением к разным файлам.
- Результаты расчетов могут сохраняться в файлы различных форматов для последующего анализа.

Описанная технология является общепринятой и не зависит от языка программирования. Как правило, для удобства работы под каждый язык создаются свои редакторы кода, где помимо обычных инструментов редактирования добавлено много специальных режимов, облегчающих написание и отладку программ.

2. Первая программа на R

В языке R файлы с командами языка называются скриптами (сценариями) и имеют расширение R (например, test12.R, exDefDeviation.R, hello.R и т.д.). По сути это обычные текстовые файлы и они могут быть открыты любым текстовым редактором. По расширению .R среда R видит «свои» файлы, быстро их загружает и запускает на выполнение. Вместе с тем, код на языке R может храниться в файле с любым расширением, в этом случае загрузка скрипта в R будет происходить дольше.

Подготовим первый скрипт и запишем в нем код программы, выводящей сообщение “Это моя первая программа!”.

Для хранения своих собственных скриптов создайте на одном из локальных дисков компьютера (c:\, d:\, ...) или сетевом диске папку для работы с R, назовите ее lessons.R и перейдите в нее. Папка должна быть пуста.

Запускаем R, выбираем меню Файл – Новый скрипт. Появляется пустое окно, в котором нужно написать строчку

```
print("Это моя первая программа!")
```

Для запуска содержимого скрипта (рис. 1) нужно выделить код и дать команду среде R на его выполнение. Ниже описана последовательность действий для случая запуска R в ОС Windows. Для других операционных систем сочетания клавиш для выполнения необходимых действий будут другими.

Нажимаем сочетание клавиш Ctrl-a (команда «Выделить все») или находим пункт меню «Правка-Выделить все», затем сочетание Ctrl-r (команда «Запустить строку или блок») или «Правка-Запустить строку или блок». В окне «R console» появится код из скрипта и строчкой ниже результат его выполнения (рис. 2).

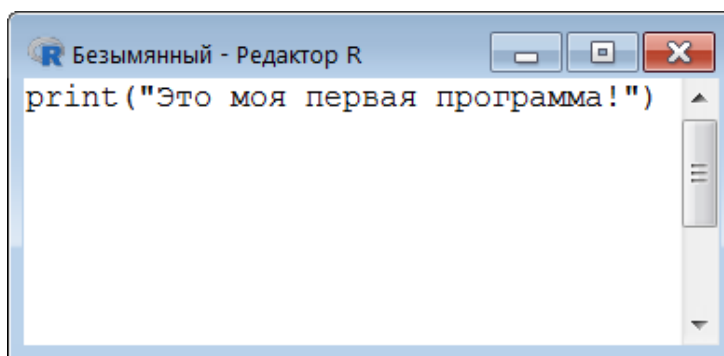
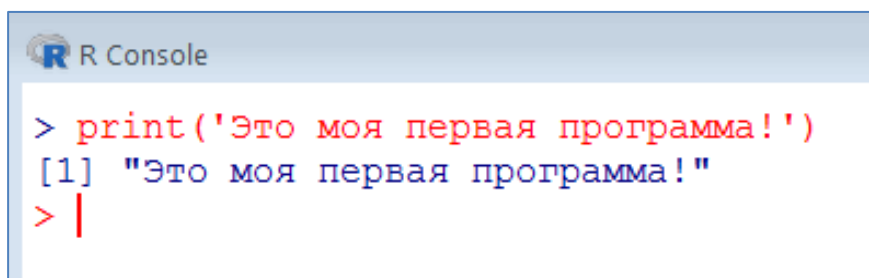


Рис. 1. Содержимое скрипта с кодом программы на языке R

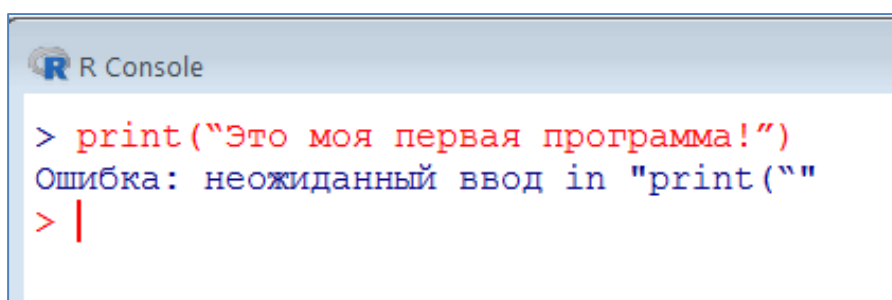
Отметим, что во всех приводимых примерах необходимо набирать код самостоятельно, вручную. При попытке копирования кода из книги будет регистрироваться ошибка (рис. 3).

Подумайте, почему?

A screenshot of the R Console window. The title bar says "R Console". The prompt ">" is followed by the command `print('Это моя первая программа!')` in red. The output is `[1] "Это моя первая программа!"` in blue. The prompt ">" is followed by a vertical bar cursor.

```
> print('Это моя первая программа!')
[1] "Это моя первая программа!"
> |
```

Рис. 2. Результат выполнения программы

A screenshot of the R Console window. The title bar says "R Console". The prompt ">" is followed by the command `print("Это моя первая программа!")` in red. The output is an error message: `Ошибка: неожиданный ввод in "print("` in blue. The prompt ">" is followed by a vertical bar cursor.

```
> print("Это моя первая программа!")
Ошибка: неожиданный ввод in "print("
> |
```

Рис. 3. Ошибка из-за некорректного копирования кода

Важно: при программировании на любом языке очень полезно использовать так называемые «горячие клавиши» — определенные клавиатурные сочетания, приводящие к различным действиям с программой. В каждой среде разработки сочетания клавиш свои, и их нужно знать и активно ими пользоваться. Использование горячих клавиш значительно ускоряет разработку и отладку программ, это гораздо быстрее, чем постоянное перемещение мышки по экрану и блуждание в многочисленных пунктах меню.

Сохраним скрипт в папку `lessons.R`. Сделать это можно тремя способами:

- щелкнуть по пиктограмме «Сохранить скрипт» в панели инструментов под строкой меню

- выбрать пункт меню «Файл–Сохранить»
- нажать Ctrl-s (рис. 4)

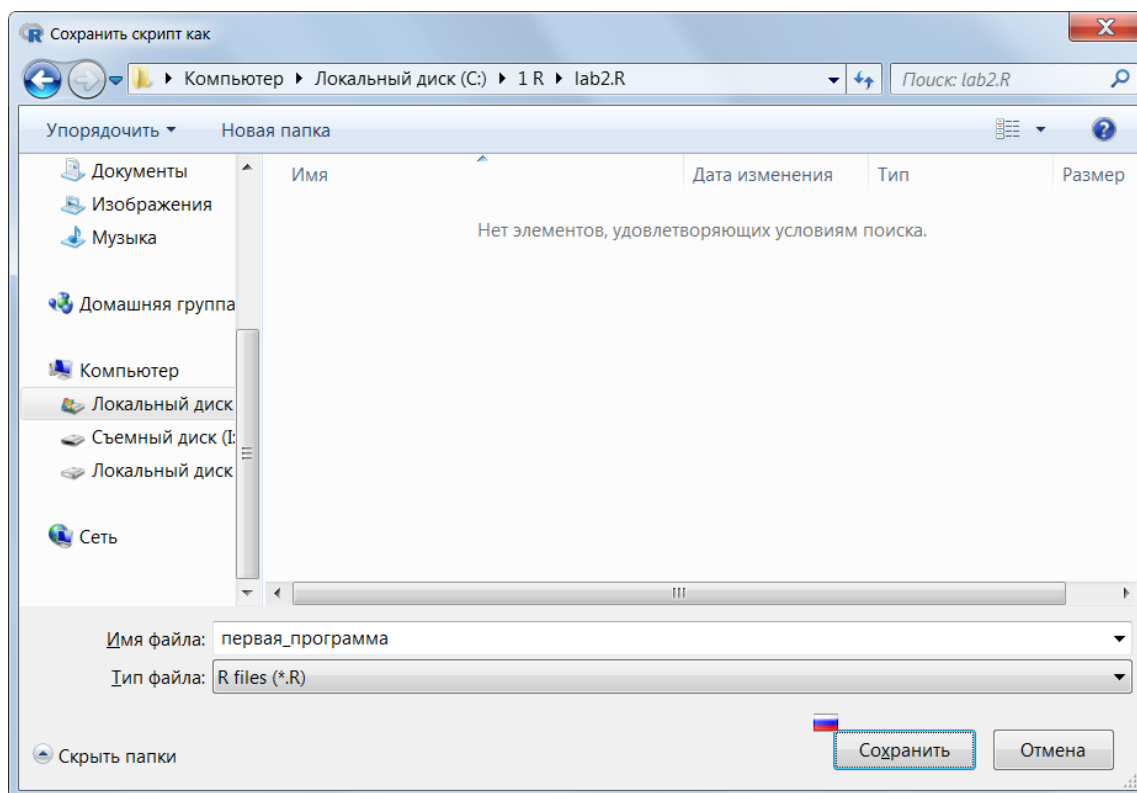


Рис. 4. Сохранение скрипта в файл

После сохранения в папке lessons.R должен появиться файл с расширением R. Для загрузки скрипта используйте пункт меню «Файл-Открыть скрипт» или соответствующую пиктограмму.

Если по каким-то причинам код был сохранен в файле с другим расширением, то для его загрузки в R нужно дополнительно указать, чтобы при выборе файла выводились файлы всех типов (см. рис. 5).

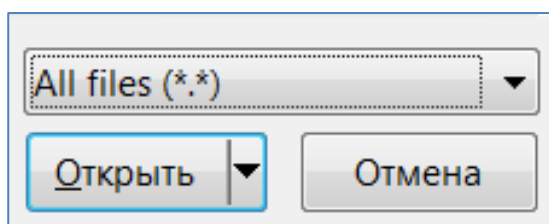


Рис. 5. Режим вывода всех файлов при загрузке скрипта

После загрузки скрипт доступен для его модификации и запуска. Можно загрузить и держать открытыми несколько разных скриптов, копировать и переносить код между ними и т.д.

Добавим в программу несколько строк. Откройте скрипт и скопируйте строчку программы несколько раз, добавив для вывода номер строки и комментариев:

```
print("Это моя первая программа! 1")    # первая строка
print("Это моя первая программа! 2")    # вторая
print("Это моя первая программа! 3")    # третья
...
print("Это моя первая программа! 10")   # последняя строчка
```

Запустите программу. На экран будут выведены все строчки. При написании программы часто бывает нужно проверить работу лишь нескольких строк кода, не запуская весь скрипт.

Задание. Разберитесь, как запустить только строки 2 и 3, запустите их.

Отдельно следует сказать об именовании файлов со скриптами: имя файла может быть на любом языке, но, при передаче скрипта за пределы России, например, в международный журнал, имя файла должно быть на английском языке. Если файл назван русскими буквами, ваш получатель увидит на своем компьютере непонятные крючки вместо имени на русском языке. Эта проблема связана с различной кодировкой символов на разных компьютерах.

Кодировка символов (другое название – кодовые страницы) – это набор чисел, которые ставятся в соответствие группе алфавитно-цифровых символов, знакам пунктуации и специальным символам.

Символы английского алфавита являются стандартными и едиными во всех кодировках, а представление символов национальных алфавитов

зависит от той кодировки, которая использовалась на компьютере. В общем случае кодировки на разных компьютерах различны.

Еще одна проблема может возникнуть при выполнении файлов R в папках, в именах которых есть пробелы или русские буквы. В некоторых случаях среда R может работать не стабильно, поэтому по возможности не используйте пробелы и названия на русском языке для папок с файлами R.

3. Корректный вывод сообщений на русском языке

Нередко встречаются ситуации, когда программа, написанная на одном компьютере, а затем запущенная на другом, вместо сообщений на русском языке выводит непонятные символы. Проблема заключается в том, что на компьютерах используются разные кодовые страницы для отображения русских букв.

Использование русского языка внутри скрипта для вывода различных сообщений возможно, но нужно быть уверенным, что русский текст набирался при установленной кодовой странице с именем UTF-8 или Windows-1251. Это универсальные кодировки для кириллицы, и при их использовании на другом компьютере сообщения из вашей программы должны отображаться корректно.

В среде R нет настроек для изменения кодировки, среда R использует кодировки, установленные в операционной системе по умолчанию. Для русскоязычной версии Windows кодировкой по умолчанию является кодировка Windows-1251.

Если для написания программы R используется сторонний редактор, нужно убедиться, что в настройках редактора установлена соответствующая кодовая страница. Так, например, для популярного редактора Notepad++ выбор кодировки выполняется через специальное меню (см. стр. 6).

Особенно актуален вопрос с кодировками, если программы разрабатываются на машинах с операционными системами, в которых языком по умолчанию является английский. В этом случае нужно самостоятельно разобраться, как установить в операционной системе нужную кодировку.

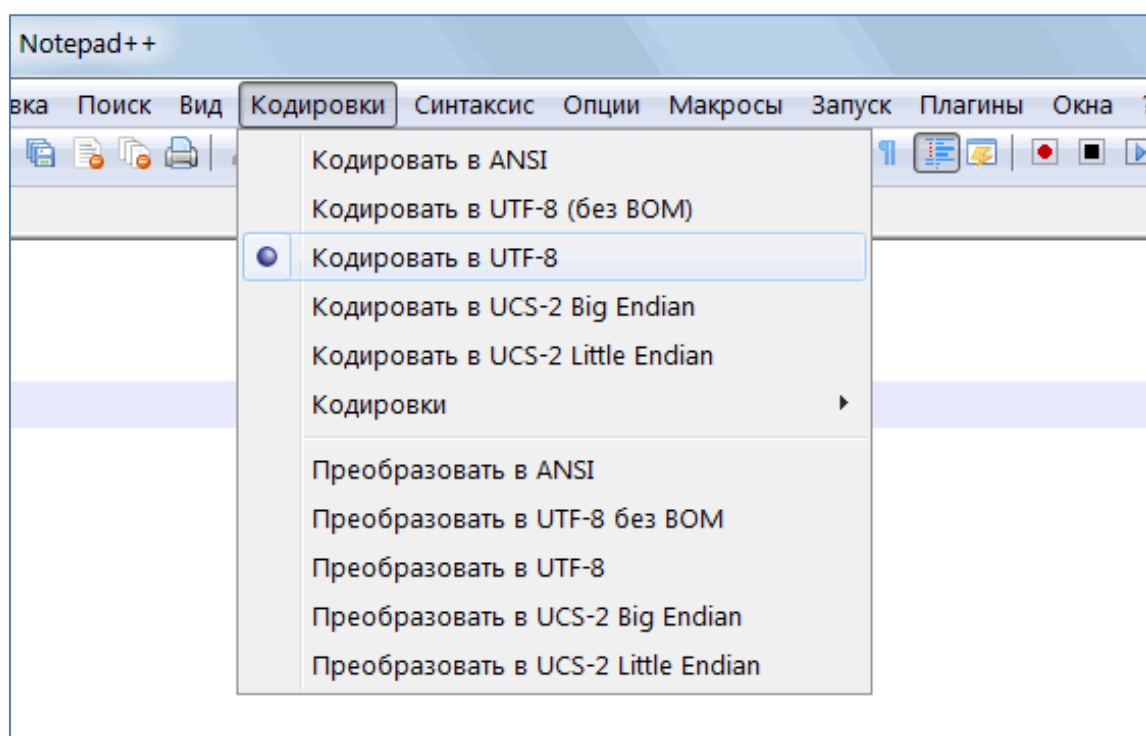


Рис. 6. Выбор кодировки в программе Notepad++

Общая рекомендация: сохранять файлы с кодом, используя кодировку UTF-8, а не Windows-1251. Причины следующие:

- Windows-1251 – набор символов и кодировка, являющаяся стандартной 8-битной кодировкой для всех русских версий Microsoft Windows
- Кодировка UTF-8 универсальная, не зависит от ОС

- Для кодирования одного символа в Windows-1251 используется 1 байт, поэтому количество возможных символов ограничено 255 (2^8-1)
- В UTF-8 один символ может кодироваться 6 байтами, сейчас используется 4. Таким образом, количество используемых в кодировке символов превышает 100000
- UTF-8 позволяет работать одновременно с несколькими языками, т.е. поддерживает тексты, в которых используются символы разных алфавитов и даже иероглифы. С использованием кодировки 1251 это невозможно
- Почти все современные веб-платформы и web-сервисы по умолчанию отображают текст через UTF-8
- UTF-8 используется для создания мультязычных проектов

4. Считывание данных с клавиатуры в программах на R

Большинство программ должно так или иначе взаимодействовать с Пользователем – реагировать на его команды, получать данные, менять алгоритм работы в зависимости от выбора Пользователя. В программах с графическим интерфейсом стандартом является взаимодействие с Пользователем через систему меню, различные окна и формы ввода.

На первом этапе для обеспечения интерактивности в своих программах нужно научиться решать более простую задачу:

- считывать ввод Пользователя с клавиатуры
- обрабатывать то, что было им введено

Для считывания данных с клавиатуры в R предназначена функция `readline()`. Разберем использование этой функции. Создайте новый скрипт, наберите код

```
name<- readline('Введите свое имя ')
```

и запустите программу. Чтобы ввести имя, нужно перейти из окна скрипта в окно консоли и там ввести имя, нажать Enter. Программа отработала, в переменной name хранится введенное имя. Как его вывести? Наберите в консоли name и нажмите Enter.

Если мы хотим после ввода имени сразу его показать, то очевидным решением было бы написать так:

```
name<- readline('Введите свое имя ')
name
```

Запустите программу. Что мы видим? Функция readline() вместо того, чтобы ждать ввода имени с клавиатуры, записала в переменную name тот код, который идет следом за вызовом этой функции. Причина такого поведения заключается в том, что выполнение программ на языке R происходит построчно, в режиме интерпретатора.

Что происходит при выполнении программы? Первая строка выполнена средой R, и функция readline() ожидает ввода. Но следом среда R выполняет следующую строчку, и именно ее и ловит readline() и помещает в переменную name.

Как заставить R работать «разумно»? Решение есть, и оно заключается в использовании фигурных скобок, говорящих среде R, что код в скобках нужно рассматривать как одну сложную команду. Добавьте в скрипт открывающую и закрывающую фигурные скобки:

```
{
  name<- readline('Введите свое имя ')
  name
}
```

Повторите выполнение программы. Теперь все работает правильно. Сделаем вывод более изящным:

```
{
  name<- readline('Введите свое имя ')
}
```

```

    print(paste("Здравствуй,", name, "!"))
}

```

Теперь после ввода имени программа здоровается с Пользователем, обращаясь к нему по имени. Что добавлено в вызов функции `print()`? Функция `print()` выводит строку, переданную как параметр в круглых скобках. Мы конструируем строку вывода, объединяя три строки в одну с помощью функции `paste()`. В функцию `paste()` передается три аргумента, разделенных запятыми:

- Первый аргумент – строка "Здравствуй,"
- Второй аргумент – имя переменной
- Третий аргумент – строка с восклицательным знаком, состоящая из одного символа.

В результате функция `paste()` конструирует одну строчку, добавляя пробелы между аргументами. Если мы не хотим добавлять пробелов, то используем похожую функцию – `paste0()`, добавив нужный пробел самостоятельно:

```

{
  name<- readline('Введите свое имя ')
  print(paste0("Здравствуй, ", name, "!"))
}

```

Чуть усложним задачу. Пусть в программе требуется регистрация Пользователя, для этого ему нужно ввести имя и фамилию. Код следующий:

```

{
  name<- readline('Введите свое имя ')
  surname<- readline('Введите свою фамилию ')
  print(paste0("Здравствуйте, ", name, " ", surname, "!"))
}

```

Последнюю сложную строчку можно разбить на две строки кода, в первой формировать строку вывода, во второй – печатать ее:

```
str4out <- paste0("Здравствуйте, ", name, " ", surname, "!" )  
print(str4out)
```

Внесите соответствующие изменения в свой скрипт и выполните его. На этом примере мы разобрались, как получать данные от Пользователя. Эти приемы будут использованы в написании более сложных программ.

Что еще можно улучшить? Обратите внимание, что итоговая строка выводится в кавычках, что не всегда нужно. Можно вывести строку без кавычек, для этого в функцию `print()` необходимо передать еще один параметр – логический переключатель, который определяет, выводить кавычки, или нет. Форма записи следующая:

```
print(str4out, quote = FALSE)
```

Разберем формальную сторону задания параметров функции подробнее.

5. Использование функций

В любом языке программирования существует огромное множество встроенных функций, которые умеют делать массу нужных и полезных действий: выводят сообщения, считывают файлы, устанавливают соединения, рисуют графики, вычисляют и анализируют значения и т.д. Программист при написании программы постоянно вызывает те или иные функции для реализации своих целей.

В R большинство важных и часто используемых функций сразу устанавливаются вместе со средой R, и их можно вызывать в своих программах. Еще большее количество специальных функций становится доступно после установки различных пакетов, расширяющих возможности языка. Кроме этого, часто сам программист пишет свои функции и затем активно их использует.

Но каким бы способом функции ни появились в среде R, все они имеют общие свойства и правила использования, о которых необходимо знать.

Формально в языке R функция отличается от переменной наличием круглых скобок: `qq` – переменная, `qq()` – функция. Круглые скобки – обязательный атрибут функции, они нужны для управления функцией, для передачи ей управляющих параметров. Другое название управляющих параметров – аргументы функции.

Использование параметров функции

Можно сказать, что круглые скобки – это двери, через которые внутрь функции попадают нужные значения: строки, числа, выражения и т.д. Функция обрабатывает эти параметры и делает что-то полезное.

Сколько аргументов может быть у функции? Сколько параметров можно ей передать? Здесь нет жестких правил, количество возможных параметров зависит от самой функции. Многие функции можно вызывать без параметров:

- функция выхода из R `quit()` или `q()`
- вызов справки `help()`
- считывание ввода с клавиатуры `readline()`
- вывод системной даты и времени `date()`

Но эти же функции вызываются и с параметрами:

- `quit('yes')`, `q('no')`, `quit('default')`
- вызов справки по конкретной функции `help(q)`, `help(help)`
- считывание ввода с клавиатуры с пояснением `readline('Ваше имя? ')`

У некоторых функций параметров не предусмотрено (напр., `date()`), некоторые функции всегда должны вызываться с параметрами (`is.numeric(...)`, `plot(...)`, `round(...)`, `sqrt(...)`, `axis(...)` и др.).

Параметры по умолчанию

Вызов функции без параметров называется вызовом функции «по умолчанию». При вызове функции без параметров функция считает, что параметры все равно переданы, но они установлены в какое-то конкретное значение «по умолчанию», и исходя из этого значения, функция работает определенным образом.

В общем случае у функции может быть множество параметров, и у параметров могут быть свои значения по умолчанию. Программист при вызове функции, как правило, задает значения лишь нескольким параметрам, оставляя значения других параметров по умолчанию. Но возникает вопрос – как функция догадается, какой из ее параметров переопределен?

Передача значений при вызове функции

Рассмотрим пример. Пусть у нашей пока не существующей и абстрактной функции `qq()` имеются три числовых параметра, и у каждого параметра есть свое значение по умолчанию. Это означает, что функцию `qq()` можно вызывать без параметров, и она что-то посчитает. Программист вызывает `qq()` и хочет переопределить второй параметр, передав значение 27. Если он запишет вызов функции в виде `qq(27)`, то функция скорее всего не догадается, что речь идет о ее втором параметре.

Существует несколько способов решения этого вопроса. Первый способ – указывать место параметра в списке всех параметров функции:

`qq(, 27)`

Смысл `(, 27)` следующий: первый параметр не меняем, оставляем пустое место, а для второго параметра передаем 27. Третий параметр и последующие, если бы они были, не трогаем.

Аналогично, для передачи 27 в третий параметр вызов функции будет таким:

`qq(, , 27)`

Такой способ не всегда удобен – чтобы им пользоваться, нужно точно знать порядковый номер нашего параметра в списке аргументов функции. У сложной функции количество аргументов может быть десятки, и в случае необходимости задать 17-й параметр легко ошибиться, переопределив вместо него 16-й или 18-й.

Поэтому более надежный и простой способ заключается в использовании имен аргументов функции и присваивании значений этим именам при вызове функции.

Суть в том, что у любой функции каждый аргумент имеет формальное имя, и если оно известно, то можно больше не думать о месте параметра в списке аргументов. При вызове функции мы просто присваиваем имени аргумента нужное значение. Вот как это работает:

Предположим, существует некоторая функция `row(x,y)` для возведения числа x в степень y (в действительности такой функции нет, название используется как пример). В результате вычислений мы должны были бы получить требуемые значения:

```
q1<- row(2,5)  # результат равен 32
```

```
q2<- row(5,2)  # результат равен 25
```

В качестве первого параметра передается x , в качестве второго – y .

Теперь вызовем функцию `row()` с явным заданием параметров, используя имена:

```
q1<- row(x=2, y=5)  # результат равен 32
```

```
q2<- row(y=5, x=2)  # результат равен 32
```

Если в первом случае, без использования имен аргументов, порядок следования параметров был важен, то во втором случае порядок не важен, мы явно определяем нужный аргумент. Если у функции есть значения по умолчанию, то мы используем такую форму вызова:

```
q1<- row(x=5)  # значение y оставляем по умолчанию
```

```
q2<- row(y=5)  # значение x оставляем по умолчанию
```


Если же у аргументов функции значений по умолчанию нет, мы получим ошибку (см. рис. 7).

```
>  
> row(x=5)  
Ошибка в row(x = 5) :аргумент "y" пропущен, умолчаний нет  
>  
> |
```

Рис. 7. Ошибка при вызове функции с отсутствием требуемого параметра

Обратите внимание, что хотя в языке R поддерживается несколько вариантов операторов присваивания, но при вызове функции с заданием значения формальному аргументу нужно использовать традиционный знак равенства (=). В этом случае мы явно определяем нужный нам аргумент требуемым значением.

Если же использовать вместо равенства присвоение в виде `y<-5`, например, `row(y<-5)`, то это приведет к следующим последствиям:

- 1) в основной программе будет создана новая переменная `y`
- 2) новой переменной `y` будет присвоено значение 5
- 3) вызов функции `row()` будет выполнен как `row(5)`, и значение 5 будет присвоено первому параметру функции `row()`, т.е. фактически выполнится `row(x=5)`.

Возникает следующий вопрос – как узнать имена аргументов функции? Вспомним, какие функции могут использоваться в программе:

1. функции, поставляемые вместе со средой R и включенные в дистрибутив R
2. функции, которые содержатся в пакетах, устанавливаемых дополнительно
3. функции, которые пишет сам Программист

В первом случае нужно вызвать подсказку, набрав ?имяФункции или help(имяФункции); во втором случае нужно смотреть документацию, которая поставляется с пакетом; в третьем случае программист сам задает имена аргументов своим функциям.

Возвращаемое значение функции

Все многообразие функций можно разделить на две большие категории:

- функции, которые возвращают значение (в результате каких-либо расчетов появляется результат)
- функции, которые делают что-то полезное (выводят сообщение, например), но никакой результат не формируют.

В некоторых языках программирования функции первого типа так и называют – функции, а функции второго типа называют процедурами.

Если результат вызова функции в дальнейшем планируется использовать, то этот результат присваивают переменной:

```
res <- pow(2,5) # в переменной res будет храниться значение 32
res.int <- round(res / pi, 0) # округлили до целого
res.dec4 <- round(32 / pi, 4) # округлили до 4 знаков после запятой
res.dec4 <- round(digits = 4, 32/ pi) # другая форма записи
```

Теперь рассмотрим использование функций, не возвращающих никакого значения. Допустим, у нас есть функция print6lines(), которая последовательно выводит на экран шесть строчек каких-либо сообщений. Если вызвать эту функцию, то она выведет на экран эти строчки. Но что будет сохранено в переменную, если ей присвоить вызов этой функции?

```
mes <- print6lines()
```

Если затем в программе вызвать переменную `mes`, то скорее всего будет выведено последнее сообщение из шести (см. рис. 8). В переменную `mes` будет записан результат команды, которая в функции `print6lines()` была последней.

```
>
> mes <- print6lines()
[1] "Это первая строка"
[1] "Это вторая строка"
[1] "А я третья!"
[1] "Я уже четвертая"
[1] "Я пятая"
[1] "А я шестая по счету, последняя!"
> mes
[1] "А я шестая по счету, последняя!"
>
```

Рис. 8. Пример работы с функцией, не возвращающей значение
Но значение переменной `mes` может быть и другим (см. рис.9):

```
>
> mes <- print6lines()
[1] "Это первая строка"
[1] "Это вторая строка"
[1] "А я третья!"
[1] "Я уже четвертая"
[1] "Я пятая"
[1] "А я шестая по счету, последняя!"
> mes
[1] -3.87905
>
```

Рис. 9. Результат функции, не возвращающей значение, неочевиден

Как понять результат, представленный на рис. 9? Почему значение переменной `mes` стало равно числу? Видимо, после вывода строк с сообщениями в функции `print6lines()` выполнялись еще какие-либо

действия, и результатом выполнения последней команды функции стало значение -3.87905.

Выше были рассмотрены случаи, когда функция или возвращает какое-либо значение, или не возвращает никакого значения. Но что делать, если в результате расчетов получен ряд значений, и все эти значения нужно сохранить? В этом случае возвращаемым значением функции должен быть вектор.

Что такое вектор в языке R, будет рассмотрено далее.

Контрольные вопросы и задания

1. Создать скрипт и перенести в него результаты самостоятельного задания № 3 из Лабораторной работы № 2: создать несколько констант и переменных разных типов, выполнить арифметические расчеты по самостоятельно подготовленным формулам с этими константами и переменными. В формулах проверить работу операторов `%%` и `% / %`.

2. Используя различные внешние программы-редакторы, подготовить в них скрипты с одним и тем же содержанием:

```
print("Проверяю работу скрипта из редактора ....")
```

где вместо многоточия вписать название программы-редактора. Использовать программы, как установленные на компьютере (Word, WordPad, блокнот и т.д.), так и найденные в интернете (AkelPad, Notepad++ и др). При сохранении скрипта проверять, где это возможно, в какой кодировке выполняется сохранение. Всегда выбирать UTF-8. После написания скрипта в редакторе открывать его в среде R и запускать на выполнение. Цель задания – сравнить удобство и возможности написания скриптов в разных редакторах.

3. Используя скрипты, написать программы, в которых требуется

- получить вводом с клавиатуры два числа
- первое число возвести в степень второго, вывести эти числа и результат
- затем второе число возвести в степень первого, вывести результат
- разделить первое число на второе, вывести результат
- выполнить деление на ноль, посмотреть на результат

Указание. Для ввода строки с клавиатуры использовать функцию `readline()`. Для преобразования строки в число использовать функции `as.integer()`, `as.double()`. Для формирования строки с результатами использовать функции `paste()`, `paste0()`. Для вывода результатов функцию `print()`.

4. Для задания № 4 из Лабораторной работы № 2 написать программы, в которых Пользователь с клавиатуры вводит значения двух переменных разных типов, которые затем сравниваются между собой. Использовать функции `readline()`, `print()` и функции преобразования типов.

5. Изучить 5-7 любых функций языка R, исследовать передачу параметров в эти функции. Выяснить формальные имена параметров и значения параметров по умолчанию. Проверить работу этих функций, написав для этого простейшие тестирующие программы. По результатам проверки работы функций написать по каждой функции краткую справку в виде комментария в тестирующей программе.