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 № 6

Функции, написанные Пользователем

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

Москва 2020

Цель лабораторной работы – изучение особенностей построения собственных функций на языке R для дальнейшего их использования при написании сложных программ.

Функции в программировании используются для логического разбиения кода на более простые части, что позволяет легко их понимать и развивать.

Функции, написанные Пользователем (программистом), дают ему широкие возможности и дополнительную гибкость: кусок кода, который делает что-то полезное, может быть использован повторно и многократно в разных вариациях, с разными наборами данных, в различных программах.

В случае необходимости что-то исправить в работе функции все исправления делаются только в одном месте – в коде описания самой функции, а результат этих изменений автоматически проявится везде, где эта функция будет вызвана.

Когда нужно создавать свою функцию? Существует несколько правил, когда это необходимо делать:

- если написанный код будет использоваться более одного раза
- если код решает некоторую частную задачу

У разработчиков программного обеспечения есть еще одно правило, связанное с удобством работы при написании кода: весь код, с которым в данный момент работает программист, должен быть виден на экране полностью.

Большое количество строк кода и необходимость постоянно прокручивать файл вверх-вниз замедляют и усложняют работу. В этом случае часть кода или сворачивают (среда разработки делает этот код невидимым, чтобы он не занимал места на экране), или выносят в отдельную функцию.

Как правило, все описания функций хранятся в специальных файлах, так называемых библиотеках. Затем библиотека добавляется к программе, и мы можем пользоваться функциями, по мере надобности вызывая их из библиотеки для решения различных задач.

Функция может возвращать результат своей работы в виде некоторого рассчитанного значения, а может просто делать какие-либо полезные действия: производить вычисления, рисовать график, считывать данные из файла или сохранять расчетные значения в файл и т.д.

В зависимости от того, что делает функция, мы либо присваиваем какой-либо переменной результат ее работы, либо просто вызываем функцию для выполнения определенных действий.

Таким образом, порядок шагов для написания функции следующий:

1. Нужно создать новый скрипт, дать ему имя, например userFunc.R
2. В этом скрипте написать конструкцию, задающую определение функции, представленную на рис. 1
3. Придумать имя функции, определить список необходимых параметров. Список параметров указывается после имени функции в круглых скобках. В частном случае параметров у функции может и не быть, но круглые скобки после имени функции обязательны. Именно круглые скобки говорят среде R, что мы вызываем функцию, а не обращаемся к переменной.
4. Написать код тела функции, в котором обрабатываются переданные параметры и выполняются различные действия.
5. Создать или открыть скрипт, в котором будет создаваться программа, использующая созданную функцию.
6. В первых строчках этого скрипта написать оператор source(“userFunc.R”) для подключения библиотеки с описанием функции. В случае, если скрипт userFunc.R располагается не в текущей папке с создаваемой программой, при вызове

source(‘…..’) нужно указывать абсолютный или относительный путь к userFunc.R.

7. В соответствии с логикой работы разрабатываемой программы вызвать функцию, передать в нее параметры, получить ее результат.

```
funcName <- function (argument) {
      statement
}
```

Рис. 1 Конструкция языка, создающая функцию Пользователя

Возможные проблемы

Если в результате запуска скрипта с программой получаем сообщение о том, что среда R не может найти файл userFunc.R, значит, нужно изменить текущую рабочую директорию среды R на ту папку, в которой сейчас сохраняются файлы R.

Если работаем в консоли R, выполняем команду перехода в папку со своими файлами

setwd(“путь_к_своей_папке”)

где функция setwd() означает "задать рабочую директорию", а “путь_к_своей_папке” может выглядеть, например, следующим образом: “c:/Rlab/lab4”.

Если работаем в RStudio, меняем настройки: Tools – Global Options – General – Default working directory и задаем свою папку.

Если в создаваемой функции при выводе сообщений использовались русские буквы, а при вызове этой функции вместо осмысленных сообщений получаем крючки и непонятные символы, значит, имеются

проблемы с совместимостью кодировки по умолчанию среды R, или RStudio, и текущей кодировки используемой операционной системы.

Решение проблем с кодировкой может потребовать определенных усилий, поэтому в этом случае рекомендуется использовать ОДИН ОБЩИЙ ФАЙЛ (скрипт) для размещения в нем определений функций и кода программы, использующего эти функции.

Разберем на примере, как создать функцию, возводящую **x** в степень **y** и выводящую результат на экран. Создаем описание функции pow (рис. 2)

```
pow <- function(x, y){
  result <- x^y
  print(paste(x, "в степени",y, 'равно', result))
}
```

Рис. 2 Пример функции Пользователя

Описание того, какие возможны варианты вызова функции pow(), представлены на рис. 3.

Часто аргументам функции присваивают значения по умолчанию, сокращая таким образом перечень обязательных аргументов. В описании функции может выполняться инициализация параметра, который может быть пропущен при ее вызове (рис. 4).

Теперь вызов функции может быть таким:

pow(3)

Результат равняется 9.

Если же вызвать функцию как обычно, с двумя параметрами, то для второго значения будет использоваться фактическое, переданное значение:

pow(3,3)

Результат равняется 27

При вызове функции фактические аргументы (2, 8) сопоставляются с формальными аргументами (x, y) в порядке следования аргументов

x = 2; y = 8

Можно вызвать функцию, используя именованные аргументы:

```
pow(8, 2)
pow(x = 8, y = 2)
pow(y = 2, x = 8)    # изменили порядок следования аргументов
pow(x = 8, 2)
pow(2, x = 8)        # изменили порядок следования аргументов
```

При вызове функции таким образом порядок фактических аргументов не имеет значения.

ПРАВИЛО. Сначала сопоставляются все именованные аргументы, а затем оставшиеся неименованные аргументы сопоставляются в порядке их следования

Рис. 3 Различные варианты вызова pow()

```
pow <- function(x, y=2){
  result <- x^y
  print(paste(x, "в степени",y, 'равно', result))
}
```

Рис. 4 Задание значения по умолчанию для одного из параметров

Использование ключевого слова return

Ключевое слово return используется в описании функции для того, чтобы явным образом задать результат, который возвратит функция при ее вызове. При достижении return происходит возврат в основную программу в точку вызова функции со значением, указанным в круглых скобках после return:

```
return(result)      # функция возвратит значение переменной result
```

Если return не использовать, то результатом работы функции будет результат выполнения последнего оператора в теле функции (см. рис. 5).

```
nSignDefiner <- function(x) {
  if (x > 0) {
    result <- "Положительное число"
  }
  else if (x < 0) {
    result <- "Отрицательное число"
  }
  else {
    result <- "А это ноль"
  }
  result
}
```

Рис. 5 Функция без return() выполняется до последнего оператора

В описании функции конструкция return(возвращаемое выражение) может встречаться несколько раз, но выполнение функции прервется при достижении первого оператора return (см. рис. 6).

```
nSignDefiner2 <- function(x) {
  if (x > 0) {
    return("Положительное число")
  }
  else if (x < 0) {
    return("Отрицательное число")
  }
  else {
    return("А это ноль")
  }
}
```

Рис. 6 Функция будет выполняться до первого return()

Использование return сокращает время работы функции, и логика ее работы становится более понятной.

Использование констант и параметров для управления поведением функции

Еще один вариант использования параметров функции – управление выводом и представлением результатов. Рассмотрим нашу функцию pow(x,y). Сейчас эта функция выводит строчку с результатом (см. рис. 2, 4). При выполнении расчетов пользоваться такой функцией неудобно, так как нам от вызванной функции нужно число – значение x в степени y, чтобы затем использовать это число в дальнейших расчетах.

Вместе с тем, например, для отладки, может понадобиться вывод информационного сообщения с указанием переданных параметров, промежуточных расчетов и т.д.

Можно ли эти два сценария работы реализовать в одной функции? Да, можно. Для этого к списку параметров добавим еще один аргумент isDebug, принимающий логические значения TRUE или FALSE. Если передано FALSE, функция будет возвращать число, в противном случае функция выведет подробную информацию о своей работе. Вариант кода приведен на рис. 7. Обратите внимание, что для параметра isDebug задано значение по умолчанию FALSE. Смысл этого в том, чтобы в расчетах не загромождать код передачей параметра с одним и тем же значением FALSE.

В больших программах режим отладки может понадобиться в разных частях, поэтому логично сделать две глобальные константы, задающие тот или иной режим выполнения функций и частей кода. В нашу программу в самом начале нужно добавить секцию констант и определить следующие константы:

DEBUG_ON <- TRUE

DEBUG_OFF <- FALSE

При создании констант программисты придерживаются следующего соглашения: константы пишутся заглавными буквами, слова в константах отделяются друг от друга нижним подчеркиванием.

Теперь наша программа написана по всем правилам (см. рис. 8).

```
#################################
# ОПИСАНИЯ ФУНКЦИЙ

pow2 <- function(x, y, isDebug = FALSE) {
  if (isDebug) {
    return(print(paste0('pow2: x=', x, ', y=', y, ', расчет=', x ^ y)))
  } else{
    return(x ^ y)
  }
}

#################################
# ОСНОВНАЯ ПРОГРАММА
{
  # запускаем функцию в режиме отладки:
  z <- pow2(4, 6, TRUE)

  # запускаем функцию для выполнения каких-либо расчетов:
  q <- 29 * pow2(3, 7) / sqrt(12)
  q
}
```

Рис. 7. Работа функции в режимах отладки и вычислений

В заключение рассмотрим правила создания имен функций. Имя функции должно начинаться со строчной буквы и быть смысловым, состоять из нескольких английских слов, отражающих работу функции; первое слово – глагол, описывающий выполняемое действие, затем идет одно или несколько существительных, отражающих то, над чем выполняется действие. Например calcRent, checkInputVal, writeTabToServer и т.д. В языке R часто в качестве разделителя в именах функций и

переменных используется точка: calc.rent, check.input.val, write.tab.to.server. В этом случае заглавные буквы не ставят.

```
################################
# КОНСТАНТЫ
DEBUG_ON <- TRUE
DEBUG_OFF <- FALSE


################################
# ОПИСАНИЯ ФУНКЦИЙ

pow2 <- function(x, y, isDebug = DEBUG_OFF) {
  if (isDebug) {
    return(print(paste0('pow2: x=', x, ', y=', y, ', расчет=', x ^ y)))
  } else{
    return(x ^ y)
  }
}

################################
# ОСНОВНАЯ ПРОГРАММА
{
  # запускаем функцию в режиме отладки:
  z <- pow2(4, 6, DEBUG_ON)

  # запускаем функцию для выполнения каких-либо расчетов:
  q <- 29 * pow2(3, 7) / sqrt(12)
  q
}
```

Рис. 8. Использование констант для удобства управления программой

Самостоятельные задания

Задание 1.

Аналогично примеру, разобранному выше, создать функцию, в которой переданный параметр x возводится в степень y, а затем делится на параметр z. Результат выводится на экран. В случае деления на ноль функция должна выводить сообщение об ошибке в параметре z.

Также функция должна проверять тип переданных параметров. Если это не целые или действительные числа, выдавать соответствующее предупреждение.

Задание 2.

Создать функцию, которая по переданному параметру возвращает наименование дня недели на русском языке. Если переданное число больше 7, делить на 7 и определять день недели по остатку от деления. Если число меньше 1, функция возвращает пробел. Если число действительное, округлять до целого.

Задание 3.

Развитие задания 2. Функция должна уметь работать с языком вывода. Если передан строковый параметр «Eng», или «eng», или «English», или «english», или «англ», или «Англ», или «анг» выводить дни недели на английском языке.

Если передан строковый параметр «rus», или «ru», или «RU», или «рус», или «ру», выводить дни недели на русском языке.

Если параметр не задан или не распознан, текст выводить на русском языке.

Задание 4.

Развитие задания 3. Функция должна уметь выводить название дня недели в двух формах: полной или сокращенной в зависимости от переданного логического параметра. Если параметр не задан, выводить полное название дня недели.

Задание 5.

Аналогично заданиям 2-4 написать функцию для вывода названия месяца.