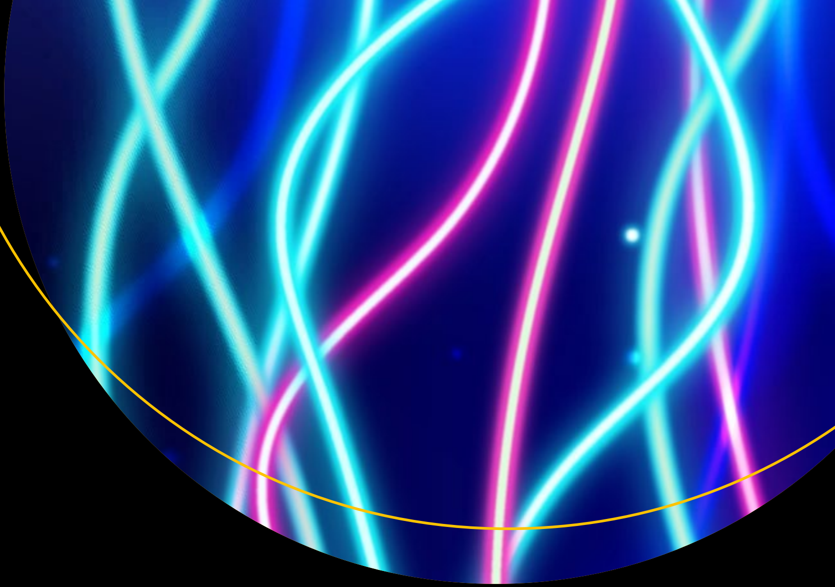

Metaprogramming in R

Advanced Statistical Programming for Data Science, Analysis and Finance

—

Thomas Mailund

Apress®

Metaprogramming in R

Advanced Statistical Programming for Data Science, Analysis and Finance

Thomas Mailund

Apress®

Metaprogramming in R: Advanced Statistical Programming for Data Science, Analysis and Finance

Thomas Mailund
Aarhus N, Denmark

ISBN-13 (pbk): 978-1-4842-2880-7 ISBN-13 (electronic): 978-1-4842-2881-4
DOI 10.1007/978-1-4842-2881-4

Library of Congress Control Number: 2017943347

Trademarked names, logos, and images may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, logo, or image we use the names, logos, and images only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

Cover image designed by Freepik

Managing Director: Welmoed Spahr
Editorial Director: Todd Green
Acquisitions Editor: Steve Anglin
Development Editor: Matthew Moodie
Technical Reviewer: Massimo Nardone
Coordinating Editor: Mark Powers
Copy Editor: Kim Wimpsett

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail rights@apress.com, or visit http://www.apress.com/rights-permissions.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at http://www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/9781484228807. For more detailed information, please visit www.apress.com/source-code.

Printed on acid-free paper

Contents at a Glance

Contents

About the Author

Thomas Mailund is an associate professor in bioinformatics at Aarhus University, Denmark. His background is in math and computer science, but for the last decade his main focus has been on genetics and evolutionary studies, particularly comparative genomics, speciation, and gene flow between emerging species.

About the Technical Reviewer

Massimo Nardone has more than 22 years of experience in security, web/mobile development, the cloud, and IT architecture. His true IT passions are security and Android.

He has been programming and teaching how to program with Android, Perl, PHP, Java, VB, Python, C/C++, and MySQL for more than 20 years.

He holds a master of science degree in computing science from the University of Salerno, Italy.

He has worked as a project manager, software engineer, research engineer, chief security architect, information security manager, PCI/SCADA auditor, and senior lead IT security/cloud/SCADA architect for many years.

He currently works as a chief information security officer (CISO) for Cargotec Oyj.

He was a visiting lecturer and supervisor for exercises at the Networking Laboratory of the Helsinki University of Technology (Aalto University), and he holds four international patents (PKI, SIP, SAML, and proxy areas).

Massimo has reviewed more than 40 IT books for different publishing companies, and he is the coauthor of *Pro Android Games* (Apress, 2015).

Introduction

Welcome to *Metaprogramming in R*. I am writing this book, and my books on R programming in general, to help make more advanced teaching material available beyond the typical introductory level most textbooks on R have. This book covers some of the more advanced techniques used in R programming such as fully exploiting functional programming, writing metaprograms (code for actually manipulating the language structures), and writing domain-specific languages to embed in R.

This book introduces metaprogramming. *Metaprogramming* is when you write programs that manipulate other programs; in other words, you treat code as data that you can generate, analyze, or modify. R is a very high-level language where all operations are functions, and all functions are data that you can manipulate.

There is great flexibility in how function calls and expressions are evaluated. The lazy evaluation semantics of R mean that arguments to functions are passed as unevaluated expressions, and these expressions can be modified before they are evaluated, or they can be evaluated in other environments than the context where a function is defined. This can be exploited to create small domain-specific languages and is a fundamental component in the "tidy verse" in packages such as `dplyr` or `ggplot2` where expressions are evaluated in contexts defined by data frames.

There is some danger in modifying how the language evaluates function calls and expressions, of course. It makes it harder to reason about code. On the other hand, adding small embedded languages for dealing with everyday programming tasks adds expressiveness to the language that far outweighs the risks of programming confusion, as long as such metaprogramming is used sparingly and in well-understood (and well-documented) frameworks.

In this book, you will learn how to manipulate functions and expressions and how to evaluate expressions in nonstandard ways. Prerequisites for reading this book are familiarity with functional programming, at least familiarity with higher-order functions, that is, functions that take other functions as an input or that return functions.

CHAPTER 1

Anatomy of a Function

Everything you do in R involves defining functions or calling functions. You cannot do any action without evaluating some function or other. Even assigning values to variables or subscripting vectors or lists involves evaluating functions. But functions are more than just recipes for how to perform different actions; they are also data objects in themselves, and there are ways of probing and modifying them.

Manipulating Functions

If you define a simple function like the following, you can examine the components it consists of:

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

There are three parts to a function: its formal parameters, its body, and the environment it is defined in. The functions `formals`, `body`, and `environment` give you these:

```
formals(f)
## $x
body(f)
## x
environment(f)
## <environment: R_GlobalEnv>
```

Formals

The formal parameters are given as a list where element names are the parameter names and values are default parameters.

T. Mailund, *Metaprogramming in R*, DOI 10.1007/978-1-4842-2881-4_1

```
g <- function(x = 1, y = 2, z = 3) x + y + z
parameters <- formals(g)
for (param in names(parameters)) {
  cat(param, "=>", parameters[[param]], "\n")
}
## x => 1
## y => 2
## z => 3
```

Strictly speaking, it is a so-called `pairlist`, but that is an implementation detail that has no bearing on how you treat it. You can treat it as if it is a `list`.

```
g <- function(x = 1, y = 2, z = 3) x + y + z
parameters <- formals(g)
for (param in names(parameters)) {
  cat(param, " => ", '"', parameters[[param]], '"', "\n", sep = "")
}
## x => "1"
## y => "2"
## z => "3"
```

For variables in this list that do not have default values, the list represents the values as the empty name. This is a special symbol that you cannot assign to, so it cannot be confused with a real value. You cannot use the `missing` function to check for a missing value in a `formals` function (that function is useful only inside a function call, and in any case there is a difference between a missing parameter and one that doesn't have a default value), but you can always check whether the value is the empty symbol.

```
g <- function(x, y, z = 3) x + y + z
parameters <- formals(g)
for (param in names(parameters)) {
  cat(param, " => ", '"', parameters[[param]], '"',
      " (", class(parameters[[param]]), ")\n", sep = "")
}
## x => "" (name)
## y => "" (name)
## z => "3" (numeric)
```

Primitive functions (those that call into the runtime system, such as `` `+` ``) do not have formals. Only functions that are defined in R.

```
formals(`+`)
## NULL
```