Programming Logic for Non-Programmers

Kat Koziar and Stephanie Labou

2025-02-04

Contents

Chapter 1

About

Bookdown reference: https://bookdown.org/yihui/bookdown/usage.html

```
BEGIN pitch()
```

Have you ever wondered how some of your colleagues can look at a computer programming script, with little prior knowledge of the language, and not only read it, but help fix the code? It's not because they know all programming languages, but because most programming languages use the same concepts and logic.

```
STRUCTURE(workshop)
```

In this interactive workshop, attendees will gain hands-on experience to understand and interpret programming logic. We will cover fundamental topics in programming including: conditional statements, loops, order of operations and logical flow, functions and arguments, and data types. Attendees will practice formulating programming arguments to accomplish common tasks, such as subsetting data based on a set of conditions.

```
WHERE prior_experience == FALSE
```

No coding experience required! Programming logic is transferable across specific languages, so learners will focus on concepts, rather than specific syntax from a specific language. Attendees will learn to interpret programming logic and build confidence to apply their understanding to various programming languages they may encounter.

```
FOR (x in example1:example5) {annotate(x)}
```

To provide real world examples of programming logic in practice, the workshop will integrate hands-on work time with examples of sample code written in R,

Python, SQL, Stata, and other languages. Attendees will practice annotating code in human understandable language and discuss the process, and any pitfalls, with their peers and the instructors.

```
IF attendee_need == "learn_programming_logic": print("register
for this workshop!")
```

1.1 Keeping this below for easy reference while we get used to the bookdown format

Prerequisites

This is a *sample* book written in **Markdown**. You can use anything that Pandoc's Markdown supports, e.g., a math equation $a^2 + b^2 = c^2$.

The **bookdown** package can be installed from CRAN or Github:

```
install.packages("bookdown")
# or the development version
# devtools::install_github("rstudio/bookdown")
```

Remember each Rmd file contains one and only one chapter, and a chapter is defined by the first-level heading `#`.

To compile this example to PDF, you need XeLaTeX. You are recommended to install TinyTeX (which includes XeLaTeX): https://yihui.org/tinytex/.

Chapter 2

Introduction

New intro for Programming Logic

2.1 Building a Mental Model

2.2 A Note on Syntax

Chapter 3

Algorithms

overview of algorithms and computer code/scripting

3.1 Breakout Activity

For this activity, you will write an algorithm to make popcorn.

Sounds simple, right? "Make popcorn" is something most people have at least a general understanding of how to do. But in this scenario, you're providing step-by-step directions for a computer to understand how to make popcorn and a computer would need to be told **every action** to take, in order to successfully make popcorn.

Your task is to write out each step needed to make popcorn.

As you write out the steps with your group, consider:

- Are you opening a container at any point?
- Are you making microwave popcorn, or using a stove?
- How much time is needed to make popcorn?
- What should the computer do if the popcorn starts burning?

Chapter 4

Loops

Drawing from Introduction to Programming Logic(Lynne O'Hanlon, 2000) Start with `for` loop as first function term "Populate an array from a for loop" as an early example; pg 376 blank table with good early exercise. Side note about infinite loop, importance of settling bounds What happens if you don't tell loop to increase? Examples of `for` loops in multiple languages (Python, C, R)? To decide: all theory and then all practice, or theory/practice/theory/practice? Also writing out result of each iteration otherwise you'll only get the last result (a common problem)

4.1 Exercise - populate an array

Let's use a `for` loop to populate an array.

We'll start with an empty array called `table` with 4 rows and columns named A, B, C, D, and E:

A	B	C	D	E

Our `for` loop:

[adapt programming logic book exercise for loop]

Grab and pen and paper (or a spreadsheet program like Google Sheets or Excel) and manually fill in the empty cells in the array.

- What is the number in cell A1?
- What is the number in cell B2?
- What is the number in cell D4?

Your filled array should look like this

A	B	C	D	E
1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20

4.2 Exercise - loop trace

Grab a pen and paper and write out the output of each iteration of the loop:

```
for (x in 0:5)
  print(x)
```

Loop output

1
2
3
4
5

4.3 Other Types of Loops

- while(check at beginning)
- do-while(check at end)
- until(check at end)

Chapter 5

Conditionals and Making Choices

Many times in programming, we want to take a certain action only if a certain condition is satisfied.

To do this, we can use conditional statements. The most commonly used format of a conditional statement in programming is an `if` statement, which is often combined with an `else` statement.

This structure tells the program to check a condition and the next step depends on whether the condition is true, or false. We can think of this as a narrative statement with conditions: "If condition A is true, do action X. If condition A is not true, do action Y."

In programming format, this narrative statement would look like:

```
IF conditionA == TRUE, do X
ELSE do Y
```

Note that `else` here is used equivalent to "if not true", meaning A == FALSE.

We are not limited to a single `TRUE/FALSE` check in an `if else` statement, where actions are limited to "x if true, y in all other scenarios".

The `else if` concept allows us to add another sequential check if the `if` statement is not true. Our updated narrative statement might be: "If condition A is true, do action X. If condition B is true, do action Z. If neither condition

A or condition B are true, do action Y."

In programming format, this updated narrative statement would look like:

```
IF conditionA == TRUE, do X
ELSE IF conditionB == TRUE, do Z
ELSE do Y
```

5.1 Boolean Operators

In programming, most things boil down to `true` or `false`. (Sometimes you may see true/false capitalized as `TRUE` and `FALSE`, but the concept is the same.)

Programming uses Boolean operators such as:

- `and` (may also see `&` used)
- `or` (may also see `|` used)
- `not` (may also see `!` to indicate negation, for instance `!=` for "not equal")
- `equals` (also `==`)
  - [SL note: most Boolean operator lists include AND, OR, NOT - should we include EQUALS here or keep as separate part of testing true/false?]

5.2 Exercise - `ifelse` trace

Let's look at some examples of conditional statements in practice.

```
if (x == y) {
  print("values are equal")
} else if (x > y) {
  print("x greater than y")
} else {
  print("x must be less than y")
}
```

For the first trace, we will set the values of `x` and `y` as:

```
x <- 37
y <- 42
```

What will be the printed output from this `ifelse` section?

Answer

x must be less than y

What if we reset `x` and `y` to:

```
x <- 75
y <- 9
```

Answer

x greater than y

5.3 Activity - if else trace

Importance of parentheses in conditionals Example of mismatched or no parentheses new lines, intendents, and other syntax standards vary by language (readability vs needs to be there to run properly)

Chapter 6

Functions

Moving into syntax What are functions, arguments in functions (order matters, explicit vs defaults) Can make own but many are prebuilt in languages

6.1 Libraries

Plus concept of libraries/packages - functions that other people have built and you can install/import Call back to popcorn - if function is make_popcorn - what is that for? What are your arguments - is it kernels or is it a bag, what time? make_popcorn(type = kernels, time = 15 minutes, butter = TRUE, salt = TRUE) vs make_popcorn() with defaults import cookbook; cookbook.make_popcorn() ### From stephanie import make_popcorn() ### From kat import make_popcorn()

6.2 Activity reprise for functions

You can think of programming libraries (or packages, or modules, depending on what term your programming language of choice uses) as similar to actual libraries. You don't need to memorize every historical event, or write your favorite novel from scratch - you can check out a book from a library to read and learn more! In the same way, your computer doesn't need to always have every single function on hand (which would take up a lot of space), it can "check out" (load) a "book" (collection of functions) created by another person. This saves computer disk space, ensures you don't have to recreate the wheel and make every function from scratch, and provides a level of standardization (e.g., everyone uses the same reference "book" so output should be the same for the same input, across users).

Libraries, packages, modules Example: min (R), max (Python), mean (SQL), regression (R, Python, Stata) Can look up documentation, most should specify arguments in each function and syntax, as well as defaults for arguments

Chapter 7

Comments and Names

Concept of comments, naming of variables and variables Briefly touch on common conventions (like `df` for dataframe); ask about disciplinary conventions for abbreviations or naming

Chapter 8

Common Issues

Some common issues = vs == (set as equal vs test for equality) Closing quotes and parentheses Overwriting variable names Order of running code matters; variable will be whatever most recently set as Spelling and capitalization matters 'X' is different from 'x' Ending a statement (needing ; or other conclusion) Direct comparison of multiple syntax, so like the same task in R, Python, C, SQL, Stata, Java You don't need to memorize specifics! Reading and writing data / files Syntax is going to be specific to a language, or package within a language Missing values may be special class, different between languages Show some examples of reading/writing data in R, Python, Stata, SQL

Chapter 9

Recap and Consultation Tips

Recap - you won't be an expert, the idea is to build up your skillset

9.1 Approaching Consultations

How you may approach consultations - prepare in advance knowing specific question, evening seeing code in advance; have student talk through their code Be clear with what you can and cannot do. Helping with programming vs statistics (for when they ask for help with interpreting something Full disclosure, I am not a statistician) Troubleshooting vs consult Ok to say you don't know! Point to documentation and learning resources

9.2 Three Areas For Errors

Code not running at all → often a syntax error

Running unexpectedly / unexpected output

input

logic

output

Chapter 10

BD Demo Introduction

You can label chapter and section titles using `{#label}` after them, e.g., we can reference Chapter 2. If you do not manually label them, there will be automatic labels anyway, e.g., Chapter **??**.

Figures and tables with captions will be placed in `figure` and `table` environments, respectively.

```
par(mar = c(4, 4, .1, .1))
plot(pressure, type = 'b', pch = 19)
```

Reference a figure by its code chunk label with the `fig:` prefix, e.g., see Figure 10.1. Similarly, you can reference tables generated from `knitr::kable()`, e.g., see Table 10.1.

```
knitr::kable(
  head(iris, 20), caption = 'Here is a nice table!',
  booktabs = TRUE
)
```

You can write citations, too. For example, we are using the **bookdown** package (Xie, 2025) in this sample book, which was built on top of R Markdown and **knitr** (Xie, 2015).

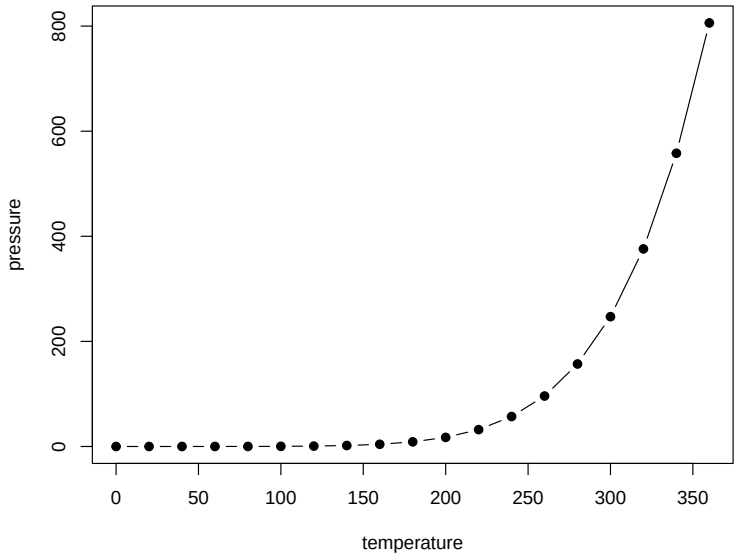

Figure 10.1: Here is a nice figure!

Table 10.1: Here is a nice table!

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa
4.6	3.4	1.4	0.3	setosa
5.0	3.4	1.5	0.2	setosa
4.4	2.9	1.4	0.2	setosa
4.9	3.1	1.5	0.1	setosa
5.4	3.7	1.5	0.2	setosa
4.8	3.4	1.6	0.2	setosa
4.8	3.0	1.4	0.1	setosa
4.3	3.0	1.1	0.1	setosa
5.8	4.0	1.2	0.2	setosa
5.7	4.4	1.5	0.4	setosa
5.4	3.9	1.3	0.4	setosa
5.1	3.5	1.4	0.3	setosa
5.7	3.8	1.7	0.3	setosa
5.1	3.8	1.5	0.3	setosa

Chapter 11

BD Demo Methods

We describe our methods in this chapter.

Math can be added in body using usual syntax like this

11.1 math example

p is unknown but expected to be around 1/3. Standard error will be approximated

$$SE = \sqrt{\frac{p(1-p)}{n}} \approx \sqrt{\frac{1/3(1-1/3)}{300}} = 0.027$$

You can also use math in footnotes like this[1].

We will approximate standard error to 0.027[2]

[1] where we mention $p = \frac{a}{b}$

[2] p is unknown but expected to be around 1/3. Standard error will be approximated

$$SE = \sqrt{\frac{p(1-p)}{n}} \approx \sqrt{\frac{1/3(1-1/3)}{300}} = 0.027$$

Bibliography

Lynne O'Hanlon (2000). *Introduction to computer programming logic*. Kendall/Hunt Pub. Co.

Xie, Y. (2015). *Dynamic Documents with R and knitr*. Chapman and Hall/CRC, Boca Raton, Florida, 2nd edition. ISBN 978-1498716963.

Xie, Y. (2025). *bookdown: Authoring Books and Technical Documents with R Markdown*. R package version 0.42.