

# A Gentle Introduction to purrr

Yingqi Jing

July 21, 2025

## Contents

1	Introduction	2
2	Select Elements	3
3	Filter Elements	3
4	Modify Elements	5
5	Combine Lists	6
6	Summarize Elements	7
7	Further Reading & Resources	8

## List of Figures

1	<a href="#">Comparison between for loops and map_dbl functions . . . . .</a>	2
---	--	---

## List of Tables

# 1 Introduction

The **purrr** package in R provides a powerful set of tools for working with lists and vectors in a functional programming style. Functions like `map()`, `map_lgl()`, `map_chr()`, `map_int()`, and `map_dbl()` allow you to iterate over elements cleanly and efficiently—offering a more readable and pipe-friendly alternative to `for` loops and `lapply()`.

For example, here is a comparison of using a `for` loop versus `purrr::map_dbl()`:

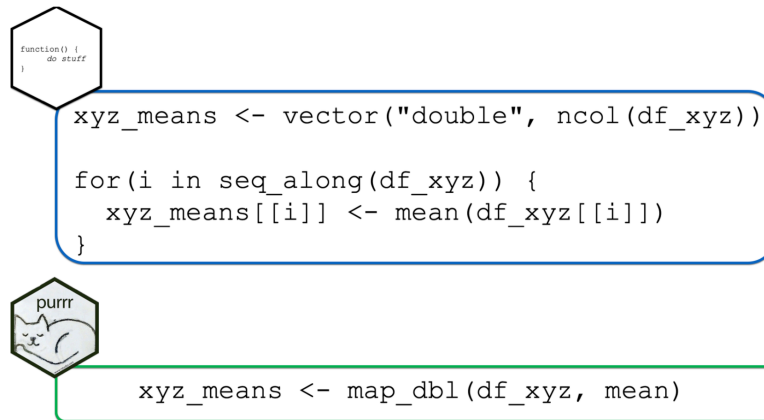


Figure 1: Comparison between `for` loops and `map_dbl` functions

The **map** family contains a number of type-specific variants. While `map()` returns a list and supports varying return types and lengths, its variants ensure a consistent output format:

Function	Output Type
<code>map()</code>	List
<code>map_dbl()</code>	Double / numeric vector
<code>map_int()</code>	Integer vector
<code>map_lgl()</code>	Logical vector
<code>map_chr()</code>	Character vector
<code>map_dfr()</code>	Data frame (row bind)
<code>map_dfc()</code>	Data frame (col bind)

## 2 Select Elements

```
list_abc <- list(a = c(1, 2), b = c(3, 4, 5), c = c("m", "n"))
```

**Example 1: Select elements by name or index**

```
# Recommended: using subset
list_abc %>% .[c("a", "b")] # or .[1:2]

# Using magrittr::extract
list_abc %>% magrittr::extract(c("a", "c"))

# Base R style
list_abc[c("a", "b")]
```

```
$a
[1] 1 2
```

```
$b
[1] 3 4 5
```

```
$a
[1] 1 2
```

```
$c
[1] "m" "n"
```

```
$a
[1] 1 2
```

```
$b
[1] 3 4 5
```

## 3 Filter Elements

**Example 2: Filter based on conditions**

```
# Keep elements with length 2
list_abc %>% keep(~ length(.x) == 2)
```

```
# Discard character vectors
list_abc %>% discard(is.character)

# Drop NULL elements
list_abc %>% append(list(d = NULL)) %>% compact()
```

```
$a
[1] 1 2
```

```
$c
[1] "m" "n"
```

```
$a
[1] 1 2
```

```
$b
[1] 3 4 5
```

```
$a
[1] 1 2
```

```
$b
[1] 3 4 5
```

```
$c
[1] "m" "n"
```

### Example 3: Slice elements inside the list

```
# First element from each
list_abc %>% map(1)

# First two elements from each
list_abc %>% map(~ .x[1:2])
```

```
$a
[1] 1
```

```
$b
[1] 3
```

```
$c
[1] "m"

$a
[1] 1 2

$b
[1] 3 4

$c
[1] "m" "n"
```

## 4 Modify Elements

### Example 4: Modify with conditions

```
# Add 1 to numeric elements
list_abc %>% keep(is.numeric) %>% modify(~ .x + 1)

# Modify if numeric, leave others unchanged
list_abc %>% modify_if(is.numeric, ~ .x + 1)

# Modify elements at positions 1 and 2
list_abc %>% modify_at(1:2, ~ .x + 10)
```

```
$a
[1] 2 3

$b
[1] 4 5 6

$a
[1] 2 3

$b
[1] 4 5 6

$c
[1] "m" "n"

$a
```

```
[1] 11 12
```

```
$b
```

```
[1] 13 14 15
```

```
$c
```

```
[1] "m" "n"
```

## 5 Combine Lists

### Example 5: Combine multiple lists

```
a <- list(a = 1:2)
b <- list(b = 3:4)
c <- list(c = 5:6)

# Append b to a
a %>% append(b)

# Prepend b to a
a %>% prepend(b)

# Splice multiple lists together
a %>%
  splice(b, c) %>%
  set_names(c("A", "B", "C")) # or use: set_names(toupper)
```

```
$a
```

```
[1] 1 2
```

```
$b
```

```
[1] 3 4
```

```
$b
```

```
[1] 3 4
```

```
$a
```

```
[1] 1 2
```

```
$A
```

```
[1] 1 2
```

```
$B
```

```
[1] 3 4
```

```
$C
```

```
[1] 5 6
```

## 6 Summarize Elements

### Example 6: Reduce to a single result

```
list_abc <- list(a = 1:2, b = 3:4, c = 5:6)
```

```
# Element-wise sum
```

```
list_abc %>% reduce(`+`)
```

```
# Element-wise multiplication
```

```
list_abc %>% reduce(`*`)
```

```
[1] 9 12
```

```
[1] 15 48
```

### Example 7: Reduce by groups

```
list_abc <- list(a = 1:2, b = 3:4, c = 5:6)
```

```
# Group-wise summation: first two together, third separately
```

```
list(1:2, 3) %>% map(~ reduce(list_abc[.x], `+`))
```

```
[[1]]
```

```
[1] 4 6
```

```
[[2]]
```

```
[1] 5 6
```

## 7 Further Reading & Resources

- [purrr Cheat Sheet \(RStudio\)](#)
- [Using the purrr Package \(r4epi\)](#)
- [purrr extras — Stanford DCL](#)
- [purrr for Parallelism — Stanford DCL](#)