Отчет по лабораторной работе 5

Тема: Функционнальное программирование (итерации и конвейеры)

Сведения о студенте

Дата: 2025-10-11
Семестр: 2 курс 1 полугодие - 3 семестр
Группа: ПИН-б-о-24-1 (2)
Дисциплина: Технологии программирования
Студент: Макаров Роман Дмитриевич

Цель работы

познакомиться с особенностями функционального программирования. Научиться применять функциональное программирования с использованием пакета `purrr`. Решить задания в соответствующем стиле программирования. Составить отчет.

Теоретическая часть

Язык R поддерживает функциональную парадигму программирования, однако для лучшей ее реализации, и для увеличения функционала, рекомендуется использовать пакет `purrr`, кроме того, необходимо также скачать и подключить пакет `repurrrsive`, содержащий наборы данных для выполнения задания №1.

Итерация

Итерация – организация обработки данных, при которой действия повторяются многократно. В программировании итерации чаще рассматриваются в качестве элементов структурного программирования, а именно как единичный шаг выполнения цикла. На практике итерации также важны и в функциональном программировании, например в качестве итерабельного процесса можно рассматривать применение одной и той же функции к разным элементам. Рассмотрим пример выполнения кода, написанного в структурном и функциональном стиле над однотипным набором данных.

Сравнение функциональной и структурной реализации:

```
# Создание тестового списка
iris_list <- as.list(iris[1:4])
# Реализация в структурном стиле
iris_mean <- list()
```

```
for (i in seq_along(iris_list)) {
iris_mean[[i]] <- mean(iris_list[[i]])
}
# Реализация в функциональном стиле через map
iris_mean <- map(iris_list, mean)
```

Варианты синтаксиса map:

```
# Реализация map без указания точки входа в функцию
iris_mean <- map(iris_list, mean)
# Реализация map с указанием точки входа в функцию
iris_mean <- map(iris_list, ~ mean(.x))
```

Второй вариант реализации позволяет показать место передачи элемента листа `(".x")` через знак тильда `("~")`. Второй способ является предпочтительнее для реализации.

Вариации функции map. map_*

Функция `map` на выходе выдает список, аналогичной структуры, однако часто необходимо сразу преобразовать вывод в некоторый вариант, для этого используются производных от функции `map`:

```
map(.x, .f, ...)
map_if(.x, .p, .f, ...)
map_at(.x, .at, .f, ...)
map_lgl(.x, .f, ...)
map_chr(.x, .f, ...)
map_int(.x, .f, ...)
map_dbl(.x, .f, ...)
map_dfr(.x, .f, ..., .id = NULL)
map_dfc(.x, .f, ...)
walk(.x, .f, ...)
```

Создание именованного числового вектора с помощью map_dbl:

```
# Создание именованного числового вектора с помощью map_dbl
iris_mean <- map_dbl(iris_list, ~ mean(.x))
```

Конвейеры (pipeline)

Конвейер – инструмент для передачи значения исходной функции в последующую. Конвейер представляет типичный элемент функционального программирования. В языке R конвейер имеет вид `%>%`.

Синтаксис конвейера:

```
function_before() %>%
function_after()
```

Выполненные задания

Задание 1

Используя тестовые данные пакета `repurrrsive` выполните следующее задание. Создайте именованный список аналогичный по структуре списку `sw_films`, для установления имени полезно использовать функцию `set_names` пакета `purrr`. В качестве имени элементов списка необходимо использовать соответствующие название фильмов (обратите внимание, что обращаться к элементам списка можно используя как индекс, так и название элемента). Выполните задание в функциональном стиле.

Задание 2

Используя документацию пакета `purrr` опишите отличия и особенности функций семейства `map_*`. Приведите примеры реализации с использование различных тестовых данных. Данные можно брать из пакета `datasets` или создав свои тестовые наборы.
Для просмотра данных из пакета `datasets` выполните код `library(help = "datasets")`

Ход работы

Задание 1.

Программный код:

```
library(repurrrsive)
library(purrr)
library(dplyr)

# Создаем именованный список на основе sw_films
named_sw_films <- sw_films %>%
  set_names(map_chr(sw_films, "title"))

# Проверяем результат
str(named_sw_films, max.level = 1)
names(named_sw_films)

# Функциональный подход с явным использованием map
named_sw_films_functional <- sw_films %>%
  map(~ .x) %>%  # Проходим по всем элементам списка
  set_names(map_chr(sw_films, ~ .x$title))  # Устанавливаем имена из заголовков

# Проверка эквивалентности
identical(named_sw_films, named_sw_films_functional)

# Демонстрация доступа к элементам по имени и индексу
```

```
# По имени фильма
named_sw_films[["A New Hope"]]
named_sw_films$`The Empire Strikes Back`

# По индексу
named_sw_films[[1]]
named_sw_films[[3]]
```

Задание 2.

*Функции семейства map_ в purrr**

Обзор семейства map_*

Функции `map_*` применяют функцию к каждому элементу входных данных и возвращают результат в определенном формате.

Основные функции и их отличия

1. `map()` - базовый вариант

Возвращает список

```
library(purrr)

# Пример с данными iris
map(iris[, 1:4], mean)  # Среднее для каждого числового столбца

# Создаем тестовые данные
test_list <- list(a = 1:3, b = 4:6, c = 7:9)
map(test_list, sum)
```

2. `map_lgl()` - возвращает логический вектор

```
# Проверяем, есть ли пропущенные значения в airquality
map_lgl(airquality, ~ any(is.na(.x)))

# Проверяем, все ли элементы > 0
numbers <- list(1:5, -1:3, 10:15)
map_lgl(numbers, ~ all(.x > 0))
```

3. `map_int()` - возвращает целочисленный вектор

```
# Количество уникальных значений в каждом столбце mtcars
map_int(mtcars, ~ length(unique(.x)))

# Длина каждого элемента
strings <- list("apple", c("banana", "cherry"), "date")
map_int(strings, length)
```

4. `map_dbl()` - возвращает числовой вектор (double)

```
# Среднее значение для каждого столбца mtcars
map_dbl(mtcars, mean)

# Стандартное отклонение
map_dbl(mtcars, sd)
```

5. `map_chr()` - возвращает вектор символов

```
# Тип данных каждого столбца в iris
map_chr(iris, class)

# Первый элемент каждого вектора
map_chr(iris, ~ as.character(.x[1]))
```

6. `map_dfr()` и `map_dfc()` - комбинирование в data frame

```
# По строкам (row-wise)
stats_by_col <- map_dfr(mtcars, ~ {
  data.frame(mean = mean(.x), sd = sd(.x), min = min(.x))
}, .id = "column")

# По столбцам (column-wise)
list_of_dfs <- list(mtcars[1:3,], mtcars[4:6,], mtcars[7:9,])
combined_df <- map_dfc(list_of_dfs, ~ .x[, 1, drop = FALSE])
```

Специализированные варианты

7. `map2()` - для двух аргументов

```
# Создаем тестовые данные
x <- list(1, 2, 3)
y <- list(10, 20, 30)

# Сумма соответствующих элементов
map2_dbl(x, y, ~ .x + .y)
```

```
# Более сложный пример с данными trees
height <- list(trees$Height[1:5], trees$Height[6:10])
volume <- list(trees$Volume[1:5], trees$Volume[6:10])
map2_dbl(height, volume, ~ cor(.x, .y))
```

8. `pmap()` - для множества аргументов

```
# Работа с параметрами распределений
params <- list(
  list(n = 5, mean = 0, sd = 1),
  list(n = 5, mean = 10, sd = 2),
  list(n = 5, mean = -5, sd = 0.5)
)

pmap(params, rnorm)
```

9. `map_if()` и `map_at()` - условное применение

```
# Применяем функцию только к числовым столбцам
map_if(iris, is.numeric, scale)

# Применяем к конкретным столбцам
map_at(mtcars, c("mpg", "wt"), ~ .x * 100)
```

Примеры с реальными данными

Пример 1: Анализ данных airquality

```
# Сводная статистика по месяцам
monthly_data <- split(airquality, airquality$Month)

monthly_stats <- map_dfr(monthly_data, ~ {
  data.frame(
    month = unique(.x$Month),
    mean_temp = mean(.x$Temp, na.rm = TRUE),
    mean_ozone = mean(.x$Ozone, na.rm = TRUE),
    complete_cases = sum(complete.cases(.x))
  )
}, .id = "month_group")
```

Пример 2: Обработка текстовых данных

```
# Создаем тестовые текстовые данные
texts <- list(
  "Hello world this is a test",
```

```
  "Another example sentence here",
  "Purrr package is very useful"
)

# Статистика по текстам
text_stats <- map_dfr(texts, ~ {
  words <- strsplit(.x, " ")[[1]]
  data.frame(
    n_chars = nchar(.x),
    n_words = length(words),
    avg_word_length = mean(nchar(words))
  )
})
```

Пример 3: Работа с вложенными структурами

```
# Сложная структура данных
nested_data <- list(
  group1 = list(a = 1:3, b = 4:6),
  group2 = list(a = 7:9, b = 10:12),
  group3 = list(a = 13:15, b = 16:18)
)

# Сумма по всем элементам 'a'
map_dbl(nested_data, ~ sum(.x$a))

# Глубокое mapping
deep_sum <- map_dbl(nested_data, ~ map_dbl(.x, sum))
```

Ключевые отличия

Функция	Возвращаемый тип	Использование
`map()`	список	Универсальный вариант
`map_lgl()`	logical	Логические операции
`map_int()`	integer	Подсчет, индексы
`map_dbl()`	double	Математические вычисления
`map_chr()`	character	Текстовая обработка
`map_dfr()`	data frame	Комбинирование по строкам
`map_dfc()`	data frame	Комбинирование по столбцам

Преимущества перед базовым R

```
# Сравнение с базовым R
library(purrr)

# purrr подход
map_dbl(mtcars, mean)
```

```
# Базовый R подход
sapply(mtcars, mean)
```

Преимущества purrr:

- Единообразный синтаксис
- Лучшая обработка ошибок
- Поддержка формул (`~ .x + 1`)
- Предсказуемые типы возвращаемых значений

Функции семейства `map_*` обеспечивают согласованный и типобезопасный способ применения операций к коллекциям данных.
