

Отчет по лабораторным работам R

Дата: 2025-10-14 **Семестр:** 2 курс 1 семестр **Группа:** ПИН-Б-О-24-2 **Дисциплина:** Технологии программирования **Студент:** Пахомов Давид Вадимович

Лабораторная работа №1

Цель работы

Познакомиться с особенностями процедурного программирования. Решить задания в процедурном стиле. Составить отчет.

Теоретическая часть

Изучены основы метода процедурное программирование, ознакомился с базовыми терминами по типу Переменная, Функция, Процедура

Практическая часть

Выполненные задачи

- Задача 1: Написать программу, выполненную в процедурном стиле. Программа должна быть выполнена в виде псевдокода, в виде блок-схемы и на языке высокого уровня (ЯВУ) (здесь и далее, если не оговорено иное, при отсылке к ЯВУ необходимо выполнять код на языке R). Для построения блок схемы рекомендуется использовать ресурс draw.io или аналогичную программу. Построение блок схемы делается с учетом правил, содержащихся в презентации Императивное (процурное) программирование. Вариант 1 Напишите программу, рассчитывающую площадь трех фигур: квадрат, прямоугольник и круг. На входе программа запрашивает введение данных о фигурах (для квадрата – сторона, круг – радиус, прямоугольник – две стороны). На выходе программа указывает площади трех фигур и общую площадь.
- Задача 2: Задание 2 – Опишите, представленный код в виде псевдокода и ответьте на вопрос, что будет получено при передаче функции числа 7? Также реализуйте данный алгоритм на ЯВУ.

Задание 1:

Ключевые фрагменты кода

```
cat("Введите сторону квадрата: ")
a <- as.numeric(readline())

cat("Введите первую сторону прямоугольника: ")
b1 <- as.numeric(readline())

cat("Введите вторую сторону прямоугольника: ")
b2 <- as.numeric(readline())

cat("Введите радиус круга: ")
r <- as.numeric(readline())
```

```
sq <- a * a
rect <- b1 * b2
circ <- pi * r^2
total <- sq + rect + circ

cat("Площадь квадрата -", sq, "\n")
cat("Площадь прямоугольника -", rect, "\n")
cat("Площадь круга -", round(circ, 2), "\n")
cat("Общая площадь -", round(total, 2), "\n")
```

Пример работы программы

```
Введите сторону квадрата: 1
Введите первую сторону прямоугольника: 2
Введите вторую сторону прямоугольника: 3
Введите радиус круга: 4
Площадь квадрата - 1
Площадь прямоугольника - 6
Площадь круга - 50.27
Общая площадь - 57.27
```

Задание 2:

Ключевые фрагменты кода

```
n <- as.integer(readline())

res <- 1
i <- n

while (i > 0) {
  res <- res * i
  i <- i - 1
}

cat(res)
```

Пример работы программы

```
> source("D:/TestR.R", echo = TRUE)

> n <- as.integer(readline())
7

> res <- 1

> i <- n

> while (i > 0) {
+   res <- res * i
+   i <- i - 1
+ }

> cat(res)
5040
```

Тестирование

- ☒ Модульные тесты пройдены
- ☒ Интеграционные тесты пройдены
- ☒ Производительность соответствует требованиям

Выводы

1. Процедурное программирование основано на разбиении программы на функции и процедуры, что делает код понятным и структурированным.
2. Линейные и структурированные конструкции обеспечивают последовательное и логичное выполнение алгоритмов.
3. Изучены основные элементы языка программирования и принципы построения простых программ.

Ответы на контрольные вопросы

1. Особенности процедурного программирования — программа делится на процедуры и функции, выполняющиеся последовательно; основной акцент на алгоритме действий.
2. Линейная программа — это последовательное выполнение команд без ветвлений и циклов.
3. Переменная, процедура, функция — переменная хранит данные; процедура выполняет действия без возврата значения; функция выполняет действия и возвращает результат.
4. Безусловный оператор — команда, изменяющая ход программы без проверки условий (например, goto, break, return).

Приложения

- Ссылки на исходный код(https://github.com/ThisGod02/PahomovDV/tree/main/lab_1_R)

Лабораторная работа №2

Цель работы

Познакомиться с особенностями структурного программирования. Решить задания в структурном стиле. Составить отчет.

Теоретическая часть

Изучены основы метода структурного программирования. Изучены базовые термины такие как Цикл, Бесконечный цикл и так же изучена цель структурного программирования.

Практическая часть

Выполненные задачи

- Задача 1: Написать программу, выполненную в структурном стиле. Программа должна рассчитывать площадь фигур (программа должна корректно обрабатывать данные)

согласно варианту в приложении А). На вход программа запрашивает строку, если в нее введено название фигуры, то программа запрашивает необходимые параметры фигуры, если введено значение отличное от названия фигуры, то программа повторно предлагает ввести название фигуры, если пользователь не справляется с этой задачей более 3 раз подряд, то программа сообщает о некорректности действий пользователя и завершается. В случае введения корректных данных программа должна выдать ответ, а также описание хода решения. Программа должна быть выполнена в виде блок-схемы и на ЯВУ.

- Задача 2: Написать программу вычисляющую площадь неправильного многоугольника. Многоугольник на плоскости задается целочисленными координатами своих N вершин в декартовой системе. Стороны многоугольника не соприкасаются (за исключением соседних - в вершинах) и не пересекаются. Программа в первой строке должна принимать число N – количество вершин многоугольника, в последующих N строках – координаты соответствующих вершин (вершины задаются в последовательности против часовой стрелки). На выход программа должна выдавать площадь фигуры. Программа должна быть выполнена в виде блок-схемы и на ЯВУ.

Задача 1:

Ключевые фрагменты кода

```
errors <- 0

while(errors < 3){
  cat("Введите название искомой фигуры (круг, треугольник, квадрат,
прямоугольник)\n")

  vvod <- readline()

  if (errors == 3){
    cat ("Ну ты и дурак конечно...\n")
    break
  }

  if (vvod == "круг"){
    cat("Введите радиус круга: \n")
    circle_radius <- as.numeric(readline())
    circle_area <- pi * circle_radius * circle_radius
    cat("Площадь круга: ", circle_area)
  }
  else if(vvod == "квадрат"){
    cat("Длина стороны квадрата: \n")
    square_side <- as.numeric(readline())
    square_area <- square_side * square_side
    cat("Площадь квадрата:", square_area)
  }
  else if(vvod == "треугольник"){
    cat ("Введите стороны треугольника: \n")
    triangle_a <- as.numeric(readline())
    triangle_b <- as.numeric(readline())
    triangle_c <- as.numeric(readline())
    triangle_p <- (triangle_a + triangle_b + triangle_c) / 2
    triangle_area <- (triangle_p * (triangle_p - triangle_a) * (triangle_p -
triangle_b) * (triangle_p - triangle_c)) ** 1/2
```

```

    cat ("Площадь треугольника: ", triangle_area)
  }
  else if(vvod == "прямоугольника"){
    cat("Первая сторона прямоугольника: \n")
    rectangle_side1 <- as.numeric(readline())
    cat("Вторая сторона прямоугольника: \n")
    rectangle_side2 <- as.numeric(readline())
    rectangle_area <- rectangle_side1 * rectangle_side2
    cat("Площадь прямоугольника:\n", rectangle_area)
  }
  else if (vvod != "круг"){
    cat("eblan\n")
    errors <- errors + 1
  }
}

```

Пример работы программы

```

Введите название искомой фигуры (круг, треугольник, квадрат, прямоугольник)
круг
Введите радиус круга:
10
Площадь круга: 314.1593
Введите название искомой фигуры (круг, треугольник, квадрат, прямоугольник)
квадрат
Длина стороны квадрата:
10
Площадь квадрата: 100

```

Задача 2:

Ключевые фрагменты кода

```

N <- as.integer(readline())

points <- list()
for (i in 1:N) {
  coords <- strsplit(readline(), " ")[[1]]
  x <- as.numeric(coords[1])
  y <- as.numeric(coords[2])
  points[[i]] <- c(x, y)
}

area <- 0
for (i in 1:N) {
  x1 <- points[[i]][1]
  y1 <- points[[i]][2]

  next_index <- ifelse(i == N, 1, i + 1)
  x2 <- points[[next_index]][1]
  y2 <- points[[next_index]][2]

  area <- area + (x1 * y2 - x2 * y1)
}

area <- abs(area) / 2

```

```
cat(area, "\n")
```

Пример работы программы

```
3
1 4
2 1
3 2
2
```

Тестирование

- ☒ Модульные тесты пройдены
- ☒ Интеграционные тесты пройдены
- ☒ Производительность соответствует требованиям

Выводы

1. Структурное программирование обеспечивает понятность, надёжность и лёгкость сопровождения кода.
2. Использование базовых управляющих конструкций позволяет строить программы без избыточных переходов.
3. Принцип «сверху-вниз» помогает логично организовать проектирование и поэтапно разрабатывать сложные программы.

Ответы на контрольные вопросы

1. Особенности структурного программирования — программа строится из трёх базовых конструкций: следование, ветвление и цикл, без использования оператора goto.
2. Теорема Бёма – Якопини — любая вычислимая программа может быть реализована с помощью трёх структур: последовательность, ветвление и цикл.
3. Пропуск итерации и досрочный выход из цикла — выполняются с помощью операторов continue (пропуск текущей итерации) и break (выход из цикла раньше завершения).

Приложения

- Ссылки на исходный код(https://github.com/ThisGod02/PahomovDV/tree/main/lab_2_R)

Лабораторная работа №3

Цель работы

Познакомиться с особенностями объектно-ориентированного программирования. Научиться создавать собственные классы с использованием R6. Решить задания в соответствующем стиле программирования. Составить отчет.

Теоретическая часть

Изучены принципы ООП и получен опыт в использовании этих принципов

Практическая часть

Выполненные задачи

- Задача 1: Создайте дженерик, принимающий вектор, содержащий параметры фигуры и вычисляющий ее площадь. Для разных фигур создайте разные классы. В качестве метода по умолчанию дженерик должен выводить сообщение о невозможности обработки данных.
- Задача 2: Создайте генератор класса Микроволновая печь. В качестве данных класс должен содержать сведения о мощности печи (Вт) и о состоянии дверцы (открыта или закрыта). Данный класс должен обладать методами открыть и закрыть дверь микроволновки, а также методом, отвечающим за приготовление пищи. Метод, отвечающий за приготовление пищи, должен вводить систему в бездействие (используется Sys.sleep) на определенное количество времени (которое зависит от мощности печи) и после выводить сообщение о готовности пищи. Выполните создание двух объектов этого класса со значением по умолчанию и с передаваемыми значениями. Протестируйте работу этих объектов по приготовлению пищи.
- Задача 3: Создайте класс копилка. Описание структуры класса выполните из своего понимания копилки.

Задание 1:

Ключевые фрагменты кода

```
create_circle <- function(radius) {  
  structure(list(radius = radius), class = "circle")  
}  
  
create_rectangle <- function(length, width) {  
  structure(list(length = length, width = width), class = "rectangle")  
}  
  
create_triangle <- function(base, height) {  
  structure(list(base = base, height = height), class = "triangle")  
}  
  
create_square <- function(side) {  
  structure(list(side = side), class = "square")  
}  
  
calculate_S <- function(x) {  
  UseMethod("calculate_S")  
}  
  
calculate_S.default <- function(x) {  
  cat("Неизвестный тип фигуры\n")  
  return(NA)  
}  
  
calculate_S.circle <- function(x) {  
  S <- pi * x$radius^2  
  cat("Площадь круга с радиусом", x$radius, "=", round(S, 2), "\n")  
}
```

```

    return(S)
}

calculate_S.rectangle <- function(x) {
  S <- x$length * x$width
  cat("Площадь прямоугольника", x$length, "x", x$width, "=", S, "\n")
  return(S)
}

calculate_S.triangle <- function(x) {
  S <- 0.5 * x$base * x$height
  cat("Площадь треугольника с основанием", x$base, "и высотой", x$height, "=", S,
"\n")
  return(S)
}

calculate_S.square <- function(x) {
  S <- x$side^2
  cat("Площадь квадрата со стороной", x$side, "=", S, "\n")
  return(S)
}

circle1 <- create_circle(5)
rectangle1 <- create_rectangle(3, 8)
triangle1 <- create_triangle(3, 8)
square1 <- create_square(5)

calculate_S(circle1)
calculate_S(rectangle1)
calculate_S(triangle1)
calculate_S(square1)

```

Пример работы программы

```

Площадь круга с радиусом 5 = 78.54
Площадь прямоугольника 3 x 8 = 24
Площадь треугольника с основанием 3 и высотой 8 = 12
Площадь квадрата со стороной 5 = 25

```

Задание 2:

Ключевые фрагменты кода

```

Microwave <- R6Class(
  "Microwave",
  private = list(
    vt = 100,
    door_open = TRUE
  ),
  public = list(
    initialize = function(vt, door_open) {
      if(!missing(vt)) {
        private$vt <- vt
      }
      if(!missing(door_open)) {
        private$door_open <- door_open
      }
    }
  )

```



```

    }
  },
  open_door = function(){
    private$door_open <- TRUE
  },
  close_door = function(){
    private$door_open<- FALSE
  },
  cook = function(){
    if(private$door_open == TRUE){
      print("Перед началом готовки закройте дверь!")
      return(NA)
    }
    cat("Время ожидания готовки:", 1000/private$vt, "\n")
    Sys.sleep(1000/private$vt)
    print("Еда готова!")
  }
)
)
microwave1 = Microwave$new()
microwave2 = Microwave$new(
  vt = 1000,
  door_open = FALSE
)
microwave1$cook()
microwave1$close_door()
microwave1$cook()
microwave2$cook()
print(microwave1)

```

Пример работы программы

```

[1] "Перед началом готовки закройте дверь!"
Время ожидания готовки: 10
[1] "Еда готова!"
Время ожидания готовки: 1
[1] "Еда готова!"

```

Задание 3:

Ключевые фрагменты кода

```

Piggy_bank <- R6Class(
  "Piggy Bank",
  private = list(
    money = 0,
    unharmed = TRUE
  ),
  public = list(
    initialize = function(money){
      if(!missing(money)){
        private$money <- money
      }
    },
    put_money = function(money){
      if(private$unharmed){

```

```

        private$money <- private$money + money
        cat("Вы положили деньги в свинку копилку!\n")
    }
    else {
        cat("Свинка копилка разбита!\n")
    }
},
destroy_piggy = function(){
    if(private$unharmed){
        private$unharmed = FALSE
        cat("Вы разбили свинку копилку! В ней было:", private$money,
"рублей!\n")
        return(private$money)
    }
    else {
        cat("Свинка копилка разбита!\n")
    }
}
)
)

```

Пример работы программы

```

Вы положили деньги в свинку копилку!
Вы положили деньги в свинку копилку!
Вы разбили свинку копилку! В ней было: 300 рублей!

```

Тестирование

- ☒ Модульные тесты пройдены
- ☒ Интеграционные тесты пройдены
- ☒ Производительность соответствует требованиям

Выводы

1. ООП упрощает разработку за счёт структурирования кода вокруг объектов и их взаимодействия.
2. Принципы инкапсуляции, наследования и полиморфизма обеспечивают гибкость и расширяемость программ.
3. Классы R6 в R позволяют реализовать полноценную объектную модель внутри языка, приближенную к классическим ООП-подходам.

Ответы на контрольные вопросы

1. Принципы ООП по Алану Кею — всё есть объект; объекты взаимодействуют через сообщения; каждый объект хранит собственное состояние и поведение.
2. Механизмы ООП — инкапсуляция (скрытие данных), наследование (переиспользование кода) и полиморфизм (единый интерфейс для разных реализаций).
3. Основные понятия ООП — класс, объект, атрибут, метод, наследование и взаимодействие между объектами.

4. Создание и назначение дженериков — дженерики позволяют создавать функции и классы, работающие с разными типами данных без дублирования кода.
5. Создание класса в R6 — выполняется через функцию `R6::R6Class()`, где задаются поля, методы и конструктор (`initialize`).
6. Структура класса в R6 — включает имя класса, публичные и приватные поля, методы, а также функцию инициализации для задания начальных значений.

Приложения

- Ссылки на исходный код(https://github.com/ThisGod02/PahomovDV/tree/main/lab_3_R)

Лабораторная работа №4

Цель работы

Познакомиться с особенностями векторного программирования в R. Решить задания в соответствующем стиле программирования. Составить отчет.

Теоретическая часть

Изучены методы работы с векторами и понятия векторного программирования

Практическая часть

Выполненные задачи

- Задача 1: Предобработка данных. Создайте новый вектор `my_vector`, следующей строчкой:
`my_vector <- c(21, 18, 21, 19, 25, 20, 17, 17, 18, 22, 17, 18, 18, 19, 19, 27, 21, 20, 24, 17, 15, 24, 24, 29, 19, 14, 21, 17, 19, 18, 18, 20, 21, 21, 19, 17, 21, 13, 17, 13, 23, 15, 23, 24, 16, 17, 25, 24, 22)` В векторе `my_vector` отберите только те наблюдения, которые отклоняются от среднего меньше, чем на одно стандартное отклонение. Сохраните эти наблюдения в новую переменную `my_vector2`. При этом исходный вектор оставьте без изменений.
- Задача 2: Напишите функцию `get_negative_values`, которая получает на вход `dataframe` произвольного размера. Функция должна для каждой переменной в данных проверять, есть ли в ней отрицательные значения. Если в переменной отрицательных значений нет, то эта переменная нас не интересует, для всех переменных, в которых есть отрицательные значения мы сохраним их в виде списка или матрицы, если число элементов будет одинаковым в каждой переменной (смотри пример работы функции).

Задание 1:

Ключевые фрагменты кода

```
my_vector <- c(21, 18, 21, 19, 25, 20, 17, 17, 18, 22, 17, 18, 18, 19, 19, 27, 21, 20, 24, 17, 15, 24, 24, 29, 19, 14, 21, 17, 19, 18, 18, 20, 21, 21, 19, 17, 21, 13, 17, 13, 23, 15, 23, 24, 16, 17, 25, 24, 22)
medium_value <- mean(my_vector)
my_vector2 <- c()
for(i in my_vector){
```

```

    if(abs(i - medium_value) <= 1){
      my_vector2 <- c(my_vector2, i)
    }
  }
for(i in my_vector2){
  print(i)
}

```

Пример работы программы

```

[1] 19
[1] 20
[1] 19
[1] 19
[1] 20
[1] 19
[1] 19
[1] 20
[1] 19

```

Задание 2:

Ключевые фрагменты кода

```

get_negative_values = function(x) {
  result <- list()
  len = length(x[[1]])
  for (i in 1:length(x)){
    if (len == length(x[[i]])){
      res = c()
      for (j in x[[i]]) {
        if(!is.na(j) & j < 0){
          res = c(res, j)
        }
      }
      if (length(res) != 0){
        result <- append(result, list(res))
      }
    } else{
      print(length(i))
      print(len)
      cat("Число элементов в переменных не одинаковое.")
      return(NA)
    }
  }
  return(result)
}

```

```

test_data <- as.data.frame(list(V1 = c(NA, -0.5, -0.7, -8), V2 = c(-0.3, NA, -2, -1.2),
V3 = c(1, 2, 3, NA)))
print(get_negative_values(test_data))
test_data <- as.data.frame(list(V1 = c(-9.7, -10, -10.5, -7.8, -8.9), V2 = c(NA, -10.2,
-10.1, -9.3, -12.2), V3 = c(NA, NA, -9.3, -10.9, -9.8)))
print(get_negative_values(test_data))

```

Пример работы программы

```
[[1]]  
[1] -0.5 -0.7 -8.0  
  
[[2]]  
[1] -0.3 -2.0 -1.2  
  
[[1]]  
[1] -9.7 -10.0 -10.5 -7.8 -8.9  
  
[[2]]  
[1] -10.2 -10.1 -9.3 -12.2  
  
[[3]]  
[1] -9.3 -10.9 -9.8
```

Тестирование

- ☒ Модульные тесты пройдены
- ☒ Интеграционные тесты пройдены
- ☒ Производительность соответствует требованиям

Выводы

1. Векторизация делает код на R компактным и значительно повышает производительность.
2. Семейство функций `apply` облегчает обработку данных, сокращая использование циклов.
3. Понимание объектов и пользовательских функций — основа эффективной работы в R.

Ответы на контрольные вопросы

1. Векторизация — это выполнение операций сразу над целыми массивами (векторами) данных без явных циклов, что ускоряет вычисления.
2. Основные объекты языка R — векторы, матрицы, списки, фреймы данных (`data.frame`) и факторы.
3. Создание собственных функций — выполняется с помощью конструкции `function(аргументы) { тело функции }`, позволяющей переиспользовать код.
4. Векторизованные функции семейства `apply` — функции (`apply`, `lapply`, `sapply`, `tapply` и др.), которые применяют операции к элементам структур данных без явных циклов.

Приложения

- Ссылки на исходный код(https://github.com/ThisGod02/PahomovDV/tree/main/lab_4_R)

Лабораторная работа №5

Цель работы

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

Теоретическая часть

Изучено функциональное программирование и его применение

Практическая часть

Выполненные задачи

- Задача 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.
- Задача 2: Используя документацию пакета purrr опишите отличия и особенности функций семейства map_*. Приведите примеры реализации с использованием 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")

Задание 1:

Ключевые фрагменты кода

```
print(sw_films1$title)
films <- map(sw_films, ~ .x$title)
named_films <- set_names(sw_films, films)
print(named_films)
```

Пример работы программы

```
$title
[1] "A New Hope"

$`A New Hope`
$`A New Hope`$title
[1] "A New Hope"

$`A New Hope`$episode_id
[1] 4

$`A New Hope`$opening_crawl
[1] "It is a period of civil war.\r\nRebel spaceships, striking\r\nfrom a hidden
base, have won\r\ntheir first victory against\r\nthe evil Galactic
Empire.\r\n\r\nDuring the battle, Rebel\r\nspies managed to steal secret\r\nplans to
```

the Empire's ultimate weapon, the DEATH STAR, an armored space station with enough power to destroy an entire planet. Pursued by the Empire's sinister agents, Princess Leia races home aboard her starship, custodian of the stolen plans that can save her people and restore freedom to the galaxy...."

\$A New Hope`\$director

[1] "George Lucas"

\$A New Hope`\$producer

[1] "Gary Kurtz, Rick McCallum"

\$A New Hope`\$release_date

[1] "1977-05-25"

\$A New Hope`\$characters

[1] "http://swapi.co/api/people/1/" "http://swapi.co/api/people/2/"
[3] "http://swapi.co/api/people/3/" "http://swapi.co/api/people/4/"
[5] "http://swapi.co/api/people/5/" "http://swapi.co/api/people/6/"
[7] "http://swapi.co/api/people/7/" "http://swapi.co/api/people/8/"
[9] "http://swapi.co/api/people/9/" "http://swapi.co/api/people/10/"
[11] "http://swapi.co/api/people/12/" "http://swapi.co/api/people/13/"
[13] "http://swapi.co/api/people/14/" "http://swapi.co/api/people/15/"
[15] "http://swapi.co/api/people/16/" "http://swapi.co/api/people/18/"
[17] "http://swapi.co/api/people/19/" "http://swapi.co/api/people/81/"

\$A New Hope`\$planets

[1] "http://swapi.co/api/planets/2/" "http://swapi.co/api/planets/3/"
[3] "http://swapi.co/api/planets/1/"

\$A New Hope`\$starships

[1] "http://swapi.co/api/starships/2/" "http://swapi.co/api/starships/3/"
[3] "http://swapi.co/api/starships/5/" "http://swapi.co/api/starships/9/"
[5] "http://swapi.co/api/starships/10/" "http://swapi.co/api/starships/11/"
[7] "http://swapi.co/api/starships/12/" "http://swapi.co/api/starships/13/"

\$A New Hope`\$vehicles

[1] "http://swapi.co/api/vehicles/4/" "http://swapi.co/api/vehicles/6/"
[3] "http://swapi.co/api/vehicles/7/" "http://swapi.co/api/vehicles/8/"

\$A New Hope`\$species

[1] "http://swapi.co/api/species/5/" "http://swapi.co/api/species/3/"
[3] "http://swapi.co/api/species/2/" "http://swapi.co/api/species/1/"
[5] "http://swapi.co/api/species/4/"

\$A New Hope`\$created

[1] "2014-12-10T14:23:31.880000Z"

\$A New Hope`\$edited

[1] "2015-04-11T09:46:52.774897Z"

\$A New Hope`\$url

[1] "http://swapi.co/api/films/1/"

```

$`Attack of the Clones`
$`Attack of the Clones`$title
[1] "Attack of the Clones"

$`Attack of the Clones`$episode_id
[1] 2

$`Attack of the Clones`$opening_crawl
[1] "There is unrest in the Galactic Senate. Several thousand solar systems
have declared their intentions to leave the Republic. This separatist
movement, under the leadership of the mysterious Count Dooku, has made it
difficult for the limited number of Jedi Knights to maintain peace and order
in the galaxy. Senator Amidala, the former Queen of Naboo, is
returning to the Galactic Senate to vote on the critical issue of
creating an ARMY OF THE REPUBLIC to assist the overwhelmed Jedi...."

$`Attack of the Clones`$director
[1] "George Lucas"
...

```

Задание 2:

Ключевые фрагменты кода

```

numbers <- list(1:3, 4:6, 7:9)
mixed <- list(1, "hello", 3.14, TRUE)
df <- data.frame(a = 1:3, b = 4:6, c = 7:9)

map(numbers, ~ .x * 2)
map(numbers, length)

map_chr(numbers, ~ paste(.x, collapse = ", "))
map_chr(df, ~ paste("col", .x[1]))

map_dbl(numbers, mean)
map_dbl(df, sum)

map_int(numbers, length)
map_int(numbers, ~ .x[1])

map_lgl(numbers, ~ any(.x %% 2 == 0))
map_lgl(numbers, ~ all(.x > 0))

map_if(df, is.numeric, ~ .x * 2)
map_if(numbers, ~ length(.x) > 2, sum)

map_at(df, c("a", "c"), ~ .x * 10)
map_at(numbers, c(1, 3), ~ .x * 100)

deep_list <- list(
  list(a = 1:2, b = 3:4),
  list(c = 5:6, d = 7:8)
)
map_depth(deep_list, 2, sum)

```



```
map_vec(numbers, mean)
map_vec(numbers, length)
map_vec(numbers, ~ mean(.x) > 5)
```

Пример работы программы

```
> numbers <- list(1:3, 4:6, 7:9)

> mixed <- list(1, "hello", 3.14, TRUE)

> df <- data.frame(a = 1:3, b = 4:6, c = 7:9)

> map(numbers, ~ .x * 2)
[[1]]
[1] 2 4 6

[[2]]
[1] 8 10 12

[[3]]
[1] 14 16 18

> map(numbers, length)
[[1]]
[1] 3

[[2]]
[1] 3

[[3]]
[1] 3

> map_chr(numbers, ~ paste(.x, collapse = ", "))
[1] "1, 2, 3" "4, 5, 6" "7, 8, 9"

> map_chr(df, ~ paste("col ", .x[1]))
      a      b      c
"col 1" "col 4" "col 7"

> map_dbl(numbers, mean)
[1] 2 5 8

> map_dbl(df, sum)
  a  b  c
6 15 24

> map_int(numbers, length)
[1] 3 3 3

> map_int(numbers, ~ .x[1])
[1] 1 4 7
```

```

> map_lgl(numbers, ~ any(.x %% 2 == 0))
[1] TRUE TRUE TRUE

> map_lgl(numbers, ~ all(.x > 0))
[1] TRUE TRUE TRUE

> map_if(df, is.numeric, ~ .x * 2)
$a
[1] 2 4 6

$b
[1] 8 10 12

$c
[1] 14 16 18

> map_if(numbers, ~ length(.x) > 2, sum)
[[1]]
[1] 6

[[2]]
[1] 15

[[3]]
[1] 24

> map_at(df, c("a", "c"), ~ .x * 10)
$a
[1] 10 20 30

$b
[1] 4 5 6

$c
[1] 70 80 90

> map_at(numbers, c(1, 3), ~ .x * 100)
[[1]]
[1] 100 200 300

[[2]]
[1] 4 5 6

[[3]]
[1] 700 800 900

> deep_list <- list(
+   list(a = 1:2, b = 3:4),
+   list(c = 5:6, d = 7:8)
+ )

```

```
> map_depth(deep_list, 2, sum)
[[1]]
[[1]]$a
[1] 3

[[1]]$b
[1] 7

[[2]]
[[2]]$c
[1] 11

[[2]]$d
[1] 15

> map_vec(numbers, mean)
[1] 2 5 8

> map_vec(numbers, length)
[1] 3 3 3

> map_vec(numbers, ~ mean(.x) > 5)
[1] FALSE FALSE TRUE
```

Тестирование

- ☒ Модульные тесты пройдены
- ☒ Интеграционные тесты пройдены
- ☒ Производительность соответствует требованиям

Выводы

1. Язык R поддерживает функциональную парадигму программирования, где обработка данных выполняется через применение функций к наборам элементов.
2. Пакет purrr значительно расширяет функциональные возможности языка, обеспечивая удобные и безопасные итерации с помощью функций map и их вариаций (map_dbl, map_chr, и др.).
3. Использование анонимных функций с тильдой (~) делает код компактнее и нагляднее, показывая место передачи аргумента.

Приложения

- Ссылки на исходный код(https://github.com/ThisGod02/PahomovDV/tree/main/lab_5_R)

Лабораторная работа №6

Цель работы

познакомиться с особенностями грамотного программирования. Научиться применять грамотное программирование для создания динамических отчетов с использованием технологии R Markdown. Решить задания в соответствующем стиле программирования. Составить отчет.

Теоретическая часть

Изучено особенности грамотного программирования

Практическая часть

Выполненные задачи

- Задача 1: Используя технологию R Markdown создайте динамический документ с произвольными расчетами. Документ должен содержать вставки кода по типу inline и в виде чанков. В документе должно быть использовано различное форматирование. Также для оформления используйте каскадную таблицу стилей. Итоговый документ конвертируйте в html формат и представьте в отчете, соответствующие скрины.

Ключевые фрагменты кода

```
---  
title: "main"  
author: "The Fool "  
date: "2025-10-13"  
output:  
  html_document:  
    css: style.css  
---
```

Практическая часть

Создадим случайные данные и выполним несколько вычислений.

```
```{r data, echo=TRUE}  
set.seed(42)
x <- rnorm(100, mean = 50, sd = 10)
mean_x <- mean(x)
sd_x <- sd(x)
```

Температуру можно перевести из Кельвинов в Цельсии по формуле:

$C = K - 273.15$

Среднее значение выборки: `r round(mean_x, 2)`

Стандартное отклонение: `r round(sd_x, 2)`

```
hist(x,
 main = "Гистограмма случайных данных",
 xlab = "Значения X",
 col = "skyblue",
 border = "white")
```

```
Пример работы программы
```{r hist, echo=FALSE}
Пример показан в отчёт.jpg
```

Тестирование

- ☒ Модульные тесты пройдены
- ☒ Интеграционные тесты пройдены
- ☒ Производительность соответствует требованиям

Выводы

1. Освоен принцип создания динамических отчётов в RStudio с использованием технологии R Markdown, объединяющей текст, код и результаты вычислений.
2. Понято, что формат R Markdown позволяет автоматически обновлять данные и выводы при изменении исходного кода.
3. Изучена структура документа, включая YAML-заголовок, текстовые блоки и вставки R-кода, выполняемые при рендеринге.

Приложения

- Ссылки на исходный код(https://github.com/ThisGod02/PahomovDV/tree/main/lab_6_R)

Лабораторная работа №7

Цель работы

Познакомиться с особенностями параллельного программирования. Научиться применять параллельное программирование для ускорения работы программ, используя стандартный пакет `parallel`. Решить задания в соответствующем стиле программирования. Составить отчет.

Теоретическая часть

Изучено особенности параллельного программирования.

Практическая часть

Выполненные задачи

- Задача 1: Используя заранее подготовленные функции визуализируйте сведения о наиболее часто встречающихся словах из книг Джейн Остин по буквам английского алфавита. Книги, необходимые для анализа, находятся в пакете `janeaustenr`. Также для работы потребуется пакет `stringr`.
- Задача 2: Распараллельте фрагмент кода, представленный ниже, используя вычислительный кластер: `for(iter in seq_len(50))`
`resultiter <- mean_of_rnorm(10000)`

Задача 1:

Ключевые фрагменты кода

```
extract_words <- function(book_name) {
  text <- subset(austen_books(), book == book_name)$text
  str_extract_all(text, boundary("word")) %>% unlist() %>% tolower()
}

janeausten_words <- function() {
  books <- austen_books()$book %>% unique() %>% as.character()
  words <- sapply(books, extract_words) %>% unlist()
  words
}

select_words <- function(letter, words, min_length = 1) {
  min_length_words <- wordsnchar(words) >= min_length
  grep(paste0("^", letter), min_length_words, value = TRUE)
}

max_frequency <- function(letter, words, min_length = 1) {
  w <- select_words(letter, words = words, min_length = min_length)
  frequency <- table(w)
  if (length(frequency) == 0) return(NA)
  frequencywhich.max(frequency)
}

words <- janeausten_words()

freqs <- sapply(letters, max_frequency, words = words, min_length = 5)

barplot(freqs, las = 2, main = "Наиболее частые слова по буквам (>=5 букв)",
  ylab = "Частота", xlab = "Буквы алфавита", col = "lightblue")
```

Пример работы программы

Вывод программы показан в task_1.jpg

Задача 2:

Ключевые фрагменты кода

```
# Определяем функцию
mean_of_rnorm <- function(n) {
  random_numbers <- rnorm(n)
  mean(random_numbers)
}

# Определяем количество ядер
ncores <- detectCores(logical = FALSE)

# Создаём кластер
cl <- makeCluster(ncores)

# Передаём функцию на все узлы
clusterExport(cl, varlist = "mean_of_rnorm")
```

```
# --- Параллельный вариант ---
system.time({
  result_parallel <- parSapply(cl, seq_len(50), function(x) mean_of_rnorm(10000))
})

# Останавливаем кластер
stopCluster(cl)

# Проверяем результат
summary(result_parallel)

# --- Последовательный вариант ---
system.time({
  result_seq <- numeric(50)
  for (iter in seq_len(50))
    result_seq[iter] <- mean_of_rnorm(10000)
})

summary(result_seq)
```

Пример работы программы

Вывод программы показан в task_2.jpg

Тестирование

- ☒ Модульные тесты пройдены
- ☒ Интеграционные тесты пройдены
- ☒ Производительность соответствует требованиям

Выводы

1. Изучены основные подходы к организации параллельных вычислений в языке R с использованием базового пакета `parallel` и сторонних пакетов, таких как `foreach` и `future.apply`.
2. Рассмотрены два метода распараллеливания задач: разделение на независимые задачи и разделение по данным.
3. Освоены принципы работы с общей и разделённой памятью, влияющие на организацию вычислений и обмен данными между процессами.

Приложения

- Ссылки на исходный код(https://github.com/ThisGod02/PahomovDV/tree/main/lab_7_R)

Лабораторная работа №8

Цель работы

Познакомиться с особенностями визуального программирования. Научиться строить программы в визуальном стиле, с использованием языка Scratch. Составить отчет.

Теоретическая часть

Изучено особенностями визуального программирования.

Практическая часть

Выполненные задачи

- Задача 1: Создайте игру на произвольную тему. В игре должно быть не менее 3 активных спрайтов с прописанной логикой. Спрайт 1 – активный персонаж, управляемый пользователем. Спрайт 2 – персонаж, выполняющий действия вне зависимости от поведения пользователя. Спрайт 3 – персонаж (объект), меняющий поведение в зависимости от действий пользователя. В игре должна быть система уровней (минимум 2), реализованная через смену фонов.

Ключевые фрагменты кода

Пример работы программы

Чтобы проверить работу программы нужно зайти на сайт Scratch и загрузить файл представленный в lab_8_R

Тестирование

- ☒ Модульные тесты пройдены
- ☒ Интеграционные тесты пройдены
- ☒ Производительность соответствует требованиям

Выводы

1. Ознакомился с принципами визуального программирования и освоил создание программ с помощью блочной логики Scratch.
2. Научился разрабатывать интерактивные проекты, включая управление персонажами и взаимодействие между спрайтами.
3. Реализовал игру с несколькими уровнями, активными объектами и реагированием на действия пользователя.

Приложения

- Ссылки на исходный код(https://github.com/ThisGod02/PahomovDV/tree/main/lab_8_R)