МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

высшего образования

«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт цифрового развития

Кафедра информационных систем и технологий

Отчет по лабораторной работе №12.

Дисциплина: «Основы программной инженерии»

Выполнил:

Студент группы ПИЖ-б-о-22-1,

направление подготовки: 09.03.04

«Программная инженерия»

ФИО: Рядская Мария Александровна

Проверил:

Воронкин Р. А.

Ставрополь 2022

Тема: Лабораторная работа 2.9. Рекурсия в языке Python.

Цель работы: приобретение навыков по работе с рекурсивными функциями при написании программ с помощью языка программирования Python версии 3.x.

Выполнение работы:

1. Изучил теоретический материал работы.

2. Создал репозиторий на git.hub.

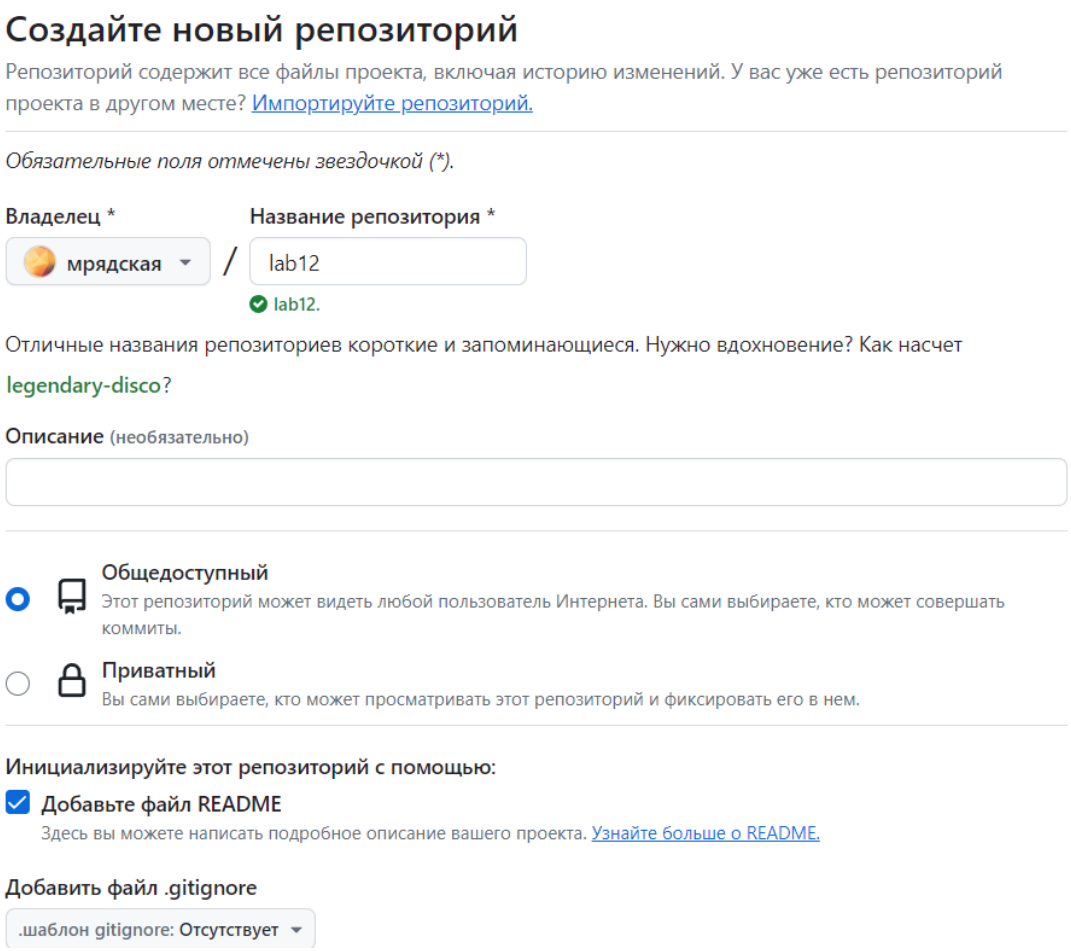
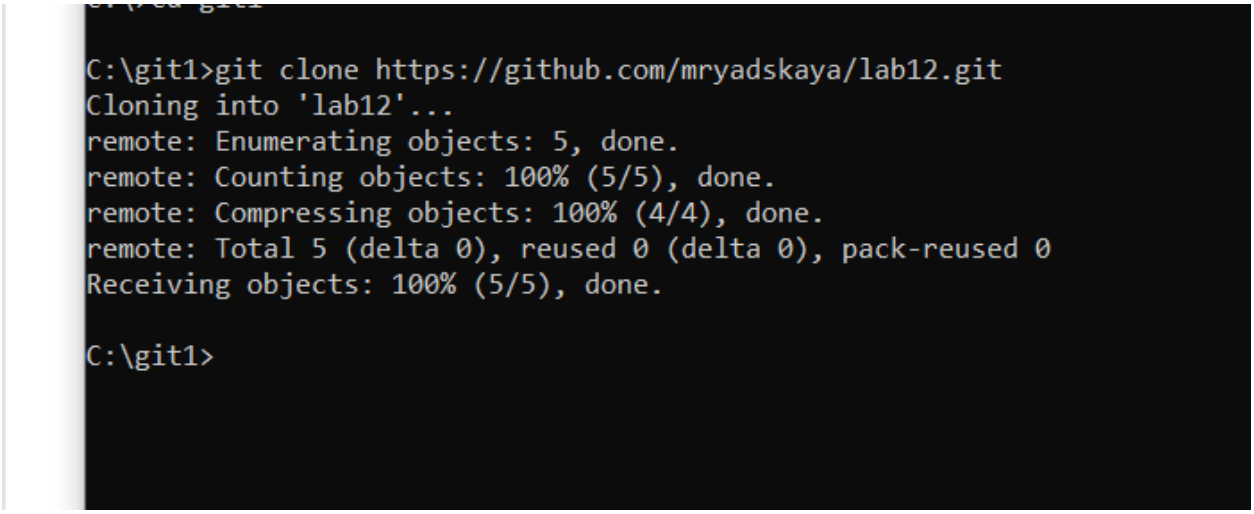


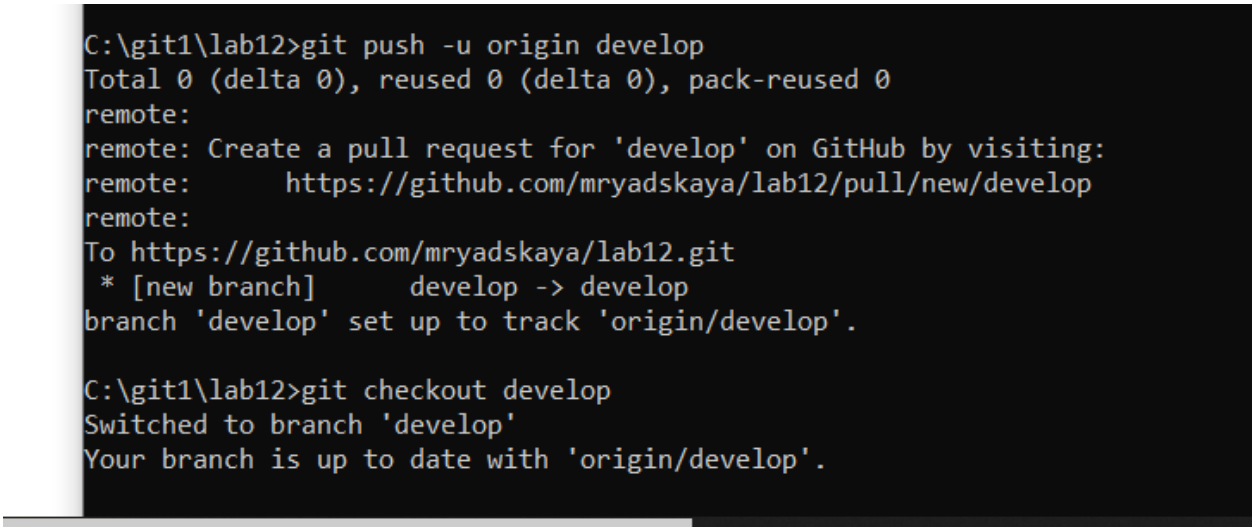
Рисунок 1 – создание репозитория

3. Клонировал репозиторий.



4. Организовать свой репозиторий в соответствии с моделью

ветвления git-flow.



6. Самостоятельно изучите работу со стандартным пакетом Python timeit .

Оцените с помощью этого модуля скорость работы итеративной и рекурсивной

версий функций factorial и fib. Во сколько раз измениться скорость работы

рекурсивных версий функций factorial и fib при использовании декоратора

lru_cache ? Приведите в отчет и обоснуйте полученные результаты.

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

from timeit import timeit
from functools import lru_cache

import sys

@lru_cache
def factorial_recursion(n):
    if n == 0:
        return 1
    elif n == 1:
        return 1
    else:
        return n * factorial_recursion(n - 1)

@lru_cache
def fib_recursion(n):
    if n == 0 or n == 1:
        return n
    else:
        return fib_recursion(n - 2) + fib_recursion(n - 1)

def factorial_iterable(n):
    product = 1
    while n > 1:
        product *= n
        n -= 1
    return product
me__ == "__main__"
```

Рисунок 5 – пример 1

```
C:\Users\ADMIN\PycharmProjects\pythonProject\venv\Scripts\python.exe "C:\git1\lab9\PyCharm\приме
Время выполнения рекурсивной функции с @lru_cache:  {7.410000034724362e-05}
Время выполнения рекурсивной функции с @lru_cache:  {0.00010179999981119181}

Process finished with exit code 0
```

Рисунок 6 – пример выполнения 1(fib_iterable)

7. Самостоятельно проработайте пример с оптимизацией хвостовых вызовов в Python. С помощью пакета timeit оцените скорость работы функций

factorial и fib с использованием интроспекции стека и без использования интроспекции стека. Приведите полученные результаты в отчет.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import sys
from timeit import timeit

class TailRecurseException(Exception):
    def __init__(self, args, kwargs):
        self.args = args
        self.kwargs = kwargs


def tail_call_optimized(g):
    """
    Эта программа показывает работу декоратора, который производит оптимизацию
    хвостового вызова. Он делает это, вызывая исключение, если оно является его
    прародителем, и перехватывает исключения, чтобы подделать оптимизацию хвоста.
    Эта функция не работает, если функция декоратора не использует хвостовой вызов.
    """

    def func(*args, **kwargs):
        f = sys._getframe()
        if (f.f_back and f.f_back.f_back and
                f.f_back.f_back.f_code == f.f_code):
            raise TailRecurseException(args, kwargs)
        else:
            while True:
                try:
                    return g(*args, **kwargs)
name__ == '__main__'
```

Рисунок 9 – код программы

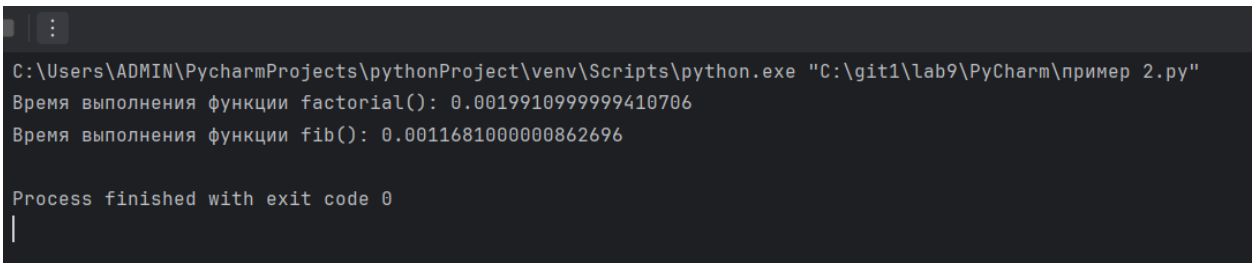


Рисунок 10 – выполнение программы

Индивидуальное задание

1 Напишите рекурсивную функцию, проверяющую правильность расстановки скобок в строке.

При правильной расстановке выполняются условия:

количество открывающих и закрывающих скобок равно.

внутри любой пары открывающая – соответствующая закрывающая скобка, скобки

расставлены правильно.

Примеры неправильной расстановки:)(, ())(, ())(() и т. п.

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-


def brackets_check(s):
    meetings = 0
    for c in s:
        if c == '(':
            meetings += 1
        elif c == ')':
            meetings -= 1
            if meetings < 0:
                return False

    return meetings == 0


if __name__ == '__main__':
    if brackets_check(input('Введите строку: ')):
        print('True')
    else:
        print('False')
```

Рисунок 11 – выполнение индивидуального задания

```
C:\Users\ADMIN\PycharmProjects\pythonProject\venv\Scripts\python.exe C:\git1\lab9\PyCharm\индивиду
Введите строку: здравствуйте) Это Рядская мария(пиж-б-о-22-1),пока(
False

Process finished with exit code 0
```

Рисунок 12 – результат выполнения индивидуального задания

9.Зафиксировал все изменения в github в ветке develop и сливание ветки develop в ветку main

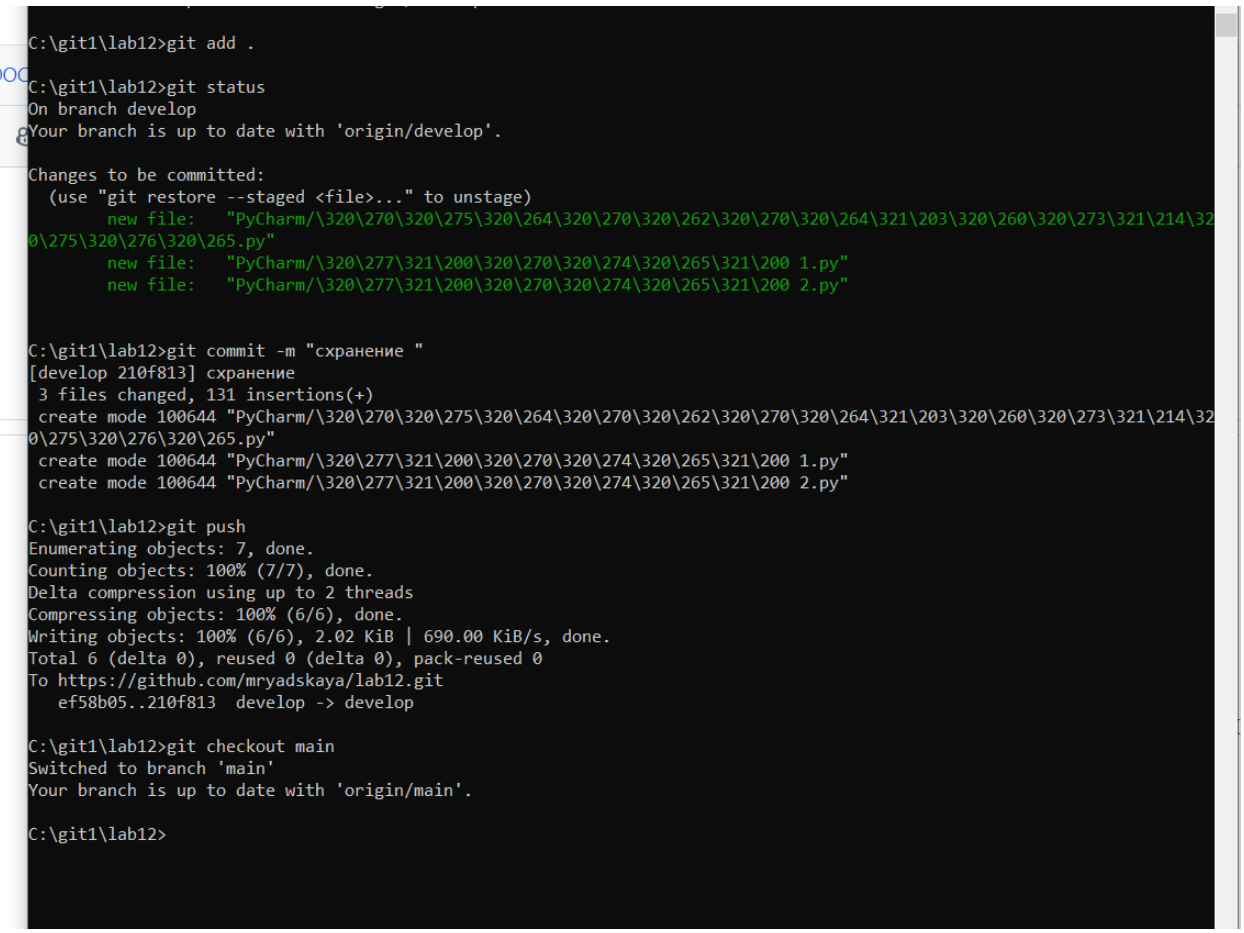


Рисунок 13 – фиксация изменений в ветку develop

Вывод: приобрел навыки по работе с рекурсивными функциями при написании программ с помощью языка программирования Python версии 3.x.

Контрольные вопросы:

1. Для чего нужна рекурсия?

Рекурсия — это прием в программировании, когда функция вызывает сама себя. Этот подход может быть использован для решения задач, которые могут

быть разбиты на более простые подзадачи. Рекурсия часто приводит к более

чистому и понятному коду, особенно в случаях, когда задача имеет структуру,

поддерживающую деление на подзадачи.

2. Что называется базой рекурсии?

Утверждение, if n == 0: return 1

3. Самостоятельно изучите что является стеком программы. Как

используется стек программы при вызове функций?

Стек программы (или стек вызовов) - это структура данных, используемая

для управления вызовами функций в программе. Каждый раз, когда функция

вызывается, информация о текущем состоянии функции (локальные переменные,

адрес возврата и т. д.) сохраняется в стеке. Когда функция завершает выполнение,

эта информация удаляется из стека, и управление возвращается к вызывающей

функции. Основные операции со стеком - это "положить" (push) и "взять" (pop).

Это относится к добавлению информации в стек при вызове функции и удалению

ее при завершении функции. Вот как происходит использование стека программы

при вызове функций:

• Когда функция вызывается, текущее состояние функции (локальные переменные, адрес возврата и т. д.) помещается в вершину стека.

• Текущая функция становится активной функцией, и управление передается в вызываемую функцию.

• Локальные переменные функции хранятся в том же фрейме стека, что и другая информация о состоянии функции.

• Эти переменные доступны только в пределах текущей функции.

• При завершении функции информация из вершины стека удаляется (pop), возвращая управление к вызывающей функции.

• Адрес возврата используется для определения, куда вернуть управление.

• Если функция вызывает саму себя (рекурсия), каждый вызов функции создает новый фрейм стека.

• Каждый уровень рекурсии имеет свои локальные переменные и адреса возврата.

Стек вызовов предоставляет эффективный механизм управления выполнением программы, обеспечивая правильный порядок вызова и возврата

функций. Он также играет важную роль в обработке исключений и управлении

памятью.

4. Как получить текущее значение максимальной глубины рекурсии в языке Python?

Чтобы проверить текущие параметры лимита, нужно запустить:

sys.getrecursionlimit()

5. Что произойдет если число рекурсивных вызовов превысит

максимальную глубину рекурсии в языке Python?

Существует предел глубины возможной рекурсии, который зависит от

реализации Python. Когда предел достигнут, возникает исключение RuntimeError

: RuntimeError: Maximum Recursion Depth Exceeded

6. Как изменить максимальную глубину рекурсии в языке Python?

Можно изменить предел глубины рекурсии с помощью вызова:

sys.setrecursionlimit(limit)

7. Каково назначение декоратора lru_cache?

Декоратор lru_cache (Least Recently Used Cache) предназначен для

кэширования результатов выполнения функций с использованием стратегии

"Наименее недавно использованный" (LRU). Он сохраняет результаты

выполнения функции для предотвращения повторных вычислений, когда те же

самые входные значения встречаются повторно.

Когда функция вызывается с определенными аргументами, lru_cache

сохраняет результат выполнения функции в кэше. Если функция вызывается с

теми же аргументами позже, она возвращает сохраненный результат, минуя

фактическое выполнение функции. Это может существенно ускорить выполнение

функций, требующих вычислений, которые могут быть дорогими по времени.

8. Что такое хвостовая рекурсия? Как проводится оптимизация

хвостовых вызовов?

Хвостовая рекурсия - это форма рекурсии, при которой рекурсивный вызов

является последней операцией в функции. Такой вызов расположен в "хвосте"

функции, то есть не сопровождается последующими операциями возврата или

обработки результатов. Важным свойством хвостовой рекурсии является то, что

она может быть оптимизирована компилятором или интерпретатором, сокращая

использование стека вызовов и уменьшая вероятность переполнения стека.

Оптимизация хвостовых вызовов, называемая также "оптимизацией

хвостовой рекурсии" или "хвостовой рекурсивной оптимизацией", заключается в

том, что компилятор или интерпретатор понимают, что после хвостового

рекурсивного вызова нет необходимости сохранять текущий контекст

выполнения (значения переменных, адрес возврата и т. д.) на стеке вызовов.