МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«СЕВЕРОКАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Кафедра

инфокоммуникаций

Институт цифрового

развития

ОТЧЁТ

по лабораторной работе №2.9

Дисциплина: «Основы программной инженерии»

Тема: «Рекурсия в языке Python»

Выполнила:

студентка 2 курса

группы Пиж-б-о-21-1

Логвинов Иван

Васильевич

Ставрополь 2022

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

1. Был создан репозиторий в Github в который были добавлены правила gitignore для работы IDE PyCharm, была выбрана лицензия MIT, сам репозиторий был клонирован на локальный сервер и был организован в соответствии с моделью ветвления git-flow.

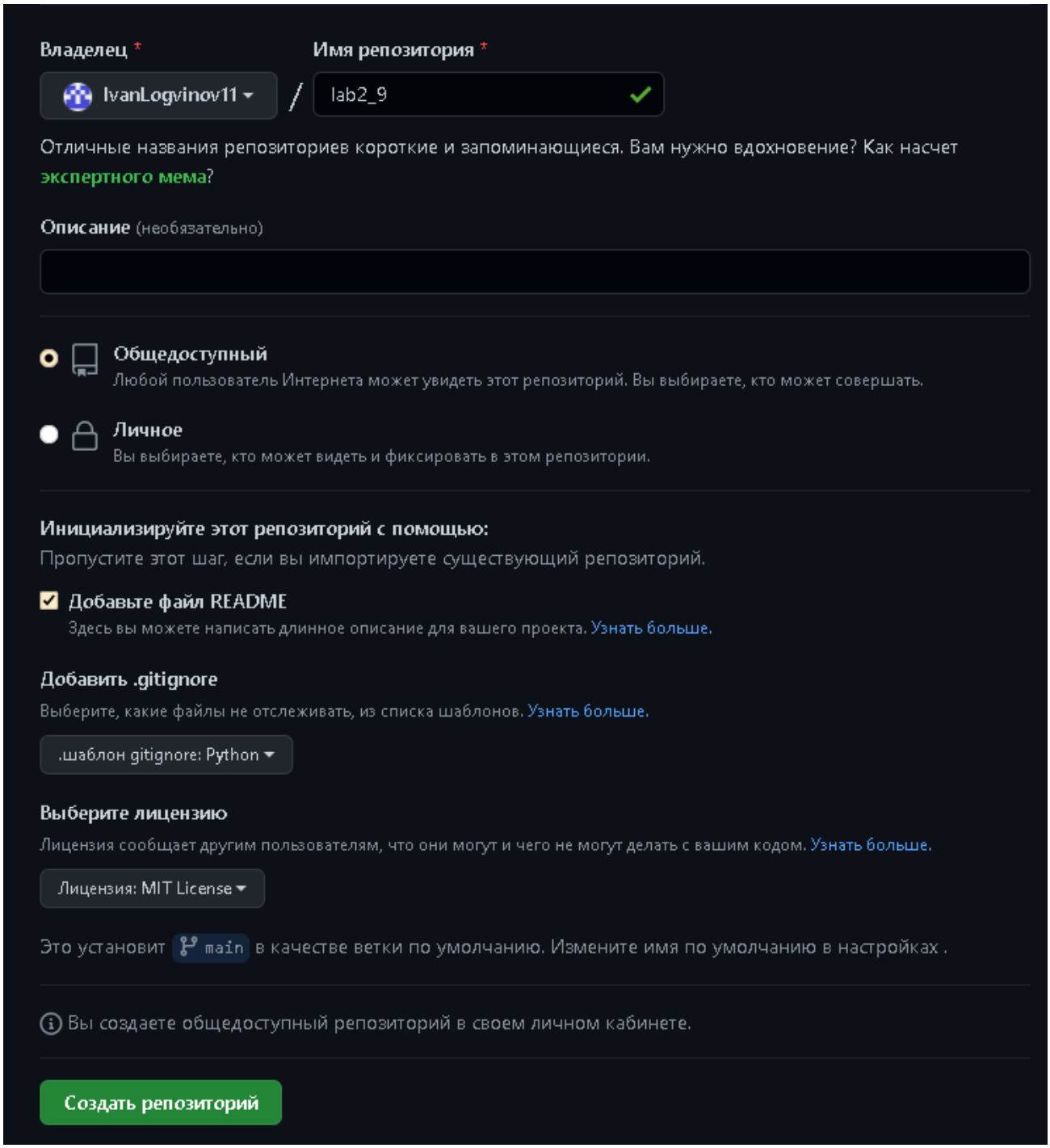

Рисунок 1 – Создание репозитория

```
Microsoft Windows [Version 10.0.18363.418]
(c) Корпорация Майкрософт (Microsoft Corporation), 2019. Все права защищены.

C:\Users\Иван>cd desktop

C:\Users\Иван\Desktop>cd гитхаб

C:\Users\Иван\Desktop\гитхаб>git clone https://github.com/IvanLogvinov11/lab2_9.git
Cloning into 'lab2_9'...
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (5/5), done.

C:\Users\Иван\Desktop\гитхаб>
```

Рисунок 2 – Клонирование репозитория

Задание №1: 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 изменится 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 -*-

import timeit
from functools import lru_cache


def factorial_iter(n):
    product = 1
    while n > 1:
        product *= n
        n -= 1
    return product


def factorial_recurse(n):
    if n == 0:
        return 1
    elif n == 1:
        return 1
    else:
        return n * factorial_recurse(n - 1)


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


if __name__ == '__main__':
```

```
    print("Time for iterative version")
    print(f'{timeit.timeit(lambda: factorial_iter(500), number=10000)},\n')
    print("Time for recurse version")
    print(f'{timeit.timeit(lambda: factorial_recurse(500), number=10000)},\n')
    print("Time for recurse_lru version")
    print(timeit.timeit(lambda: factorial_rec_lru(500), number=10000))
```

Рисунок 1 – Код задания №1

```
C:\Users\Иван\AppData\Local\Programs\Pytho
Time for iterative version
0.9382035000000001,

Time for recurse version
1.0853979,

Time for recurse_lru version
0.009941099999999917

Process finished with exit code 0
```

Рисунок 2 – Результат работы кода задания

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

import timeit
from functools import lru_cache


def fib_iter(n):
    a, b = 0, 1
    while n > 0:
        a, b = b, a + b
        n -= 1
    return a


def fib_recurse(n):
    if n == 0 or n == 1:
        return n
    else:
        return fib_recurse(n - 2) + fib_recurse(n - 1)


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

```
if __name__ == '__main__':
    print("Time for iterative version")
    print(f'{timeit.timeit(lambda: fib_iter(15), number=10000)},\n')
    print("Time for recurse version")
    print(f'{timeit.timeit(lambda: fib_recurse(15), number=10000)},\n')
    print("Time for recurse_lru version")
    print(timeit.timeit(lambda: fib_rec_lru(15), number=10000))
```

Рисунок 3 – Код задания №1 (числа Фибоначчи)

```
C:\Users\Иван\AppData\Local\Programs\Pyth
Time for iterative version
0.0092924,

Time for recurse version
1.9722971999999999,

Time for recurse_lru version
0.0008501999999999121

Process finished with exit code 0
```

Рисунок 4 – Результат работы кода задания №1

Задание №2: 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 timeit


class recursion(object):
    def __init__(self, func):
        self.func = func

    def __call__(self, *args, **kwargs):
        result = self.func(*args, **kwargs)
        while callable(result): result = result()
        return result

    def call(self, *args, **kwargs):
        return lambda: self.func(*args, **kwargs)
```

```python
@recursion
def factorial_opt(n, acc=1):
    if n == 0:
        return acc
    return factorial(n - 1, n * acc)


def factorial(n, acc=1):
    if n == 0:
        return acc
    return factorial(n - 1, n * acc)


if __name__ == '__main__':
    print("Время работы кода с использованием интроспекции")
    print(f'{timeit.timeit(lambda: factorial_opt(250), number=10000)}\n')
    print("Время работы кода без использования интроспекции")
    print(timeit.timeit(lambda: factorial(250), number=10000))
```

Рисунок 5 – Код задания №2

```
C:\Users\Иван\AppData\Local\Programs\Python\Python39\python
Время работы кода с использованием интроспекции
0.7027407

Время работы кода без использования интроспекции
0.4830491

Process finished with exit code 0
```

Рисунок 6 – Результат работы кода задания №2

Индивидуальное задание: Создайте рекурсивную функцию, печатающие все подмножества множества $\{1, 2, \ldots, N\}$.

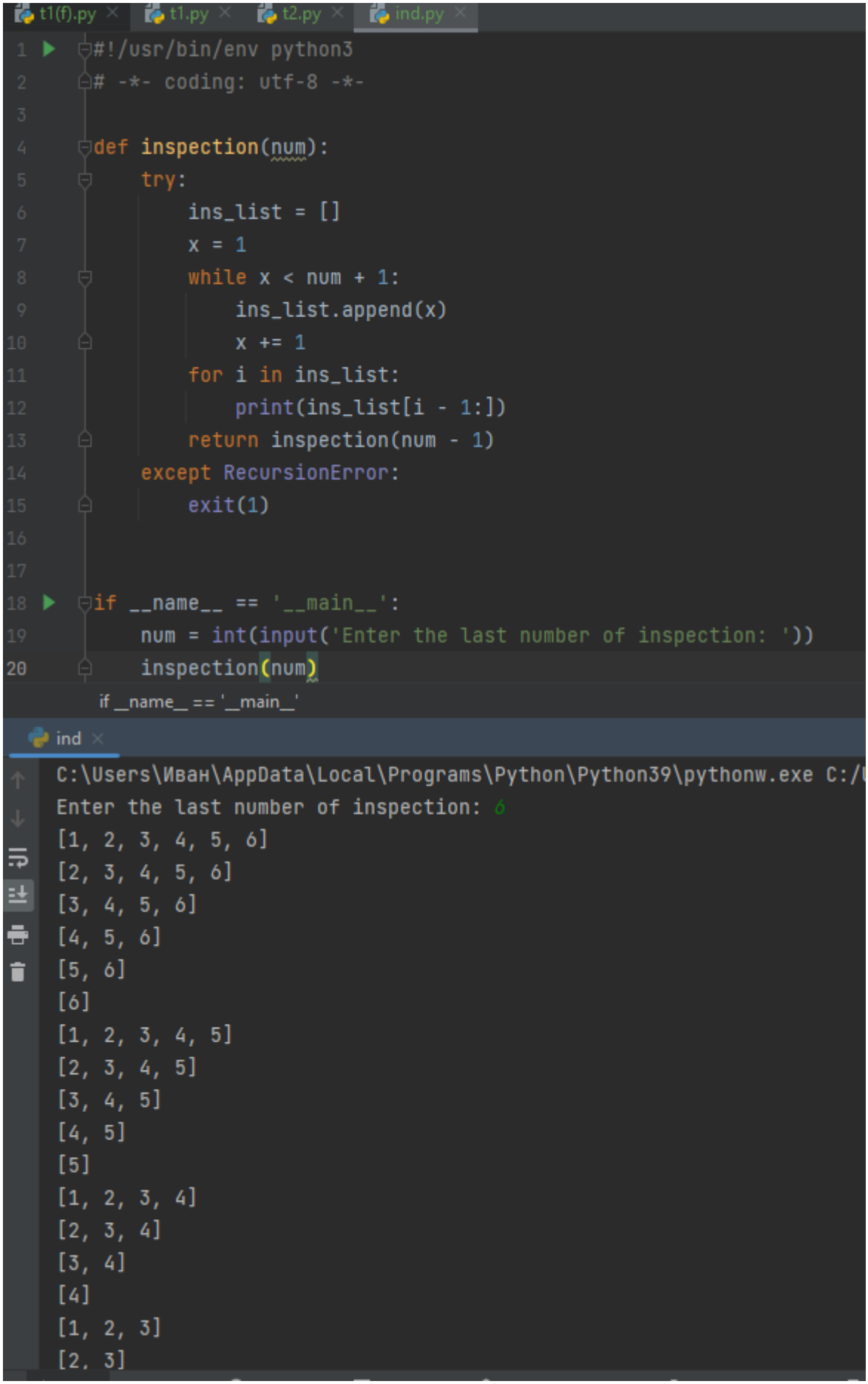

Рисунок 7 – Код и результат работы программы индивидуального задания

Контрольные вопросы

1. Для чего нужна рекурсия?

Функция может содержать вызов других функций. В том числе процедура может вызвать саму себя.

2. Что называется базой рекурсии?

У рекурсии, как и у математической индукции, есть база — аргументы, для которых значения функции определены

3. Самостоятельно изучите что является стеком программы. Как используется стек программы при вызове функций?

Максимальная глубина рекурсии ограничена движком JavaScript. Точно можно рассчитывать на 10000 вложенных вызовов, некоторые интерпретаторы допускают и больше, но для большинства из них 100000 вызовов – за пределами возможностей. Существуют автоматические оптимизации, помогающие избежать переполнения стека вызовов
(«оптимизация хвостовой рекурсии»), но они ещё не поддерживаются везде и работают только для простых случаев.

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

Функция sys. getrecursionlimit() возвращает текущее значение предела рекурсии, максимальную глубину стека интерпретатора Python.

5. Что произойдет если число рекурсивных вызовов превысит максимальную глубину рекурсии в языке Python?

Ошибка RunTimeError

6. Как изменить максимальную глубину рекурсии в языке Python?

С помощью функции setrecursionlimit() модуля sys

7. Каково назначение декоратора lru_cache ?

Декоратор @lru_cache() модуля functools оборачивает функцию с переданными в нее аргументами и запоминает возвращаемый результат соответствующий этим аргументам. Такое поведение может сэкономить время и ресурсы, когда дорогая или связанная с вводом/выводом функция периодически вызывается с одинаковыми аргументами.

8. Что такое хвостовая рекурсия? Как проводится оптимизация хвостовых вызовов?

Хвостовая рекурсия — частный случай рекурсии, при котором любой рекурсивный вызов является последней операцией перед возвратом из функции. Оптимизация хвостовой рекурсии выглядит так: