Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение

высшего образования

«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт цифрового развития

Кафедра инфокоммуникаций

**ОТЧЕТ**

**ПО ЛАБОРАТОРНОЙ РАБОТЕ №12**

**дисциплины «Основы программной инженерии»**

|  |  |  |  |
| --- | --- | --- | --- |
|  | | Выполнил:  Магомедов Имран Борисович  2 курс, группа ПИЖ-б-о-22-1,  09.03.04 «Программная инженерия», направленность (профиль) «Разработка и сопровождение программного обеспечения», очная форма обучения  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) | |
|  | | Руководитель практики:  Воронкин Р.А., кандидат технических наук, доцент кафедры инфокоммуникаций  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) | |
|  |  | |  |

Отчет защищен с оценкой \_\_\_\_\_\_\_\_\_ Дата защиты\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

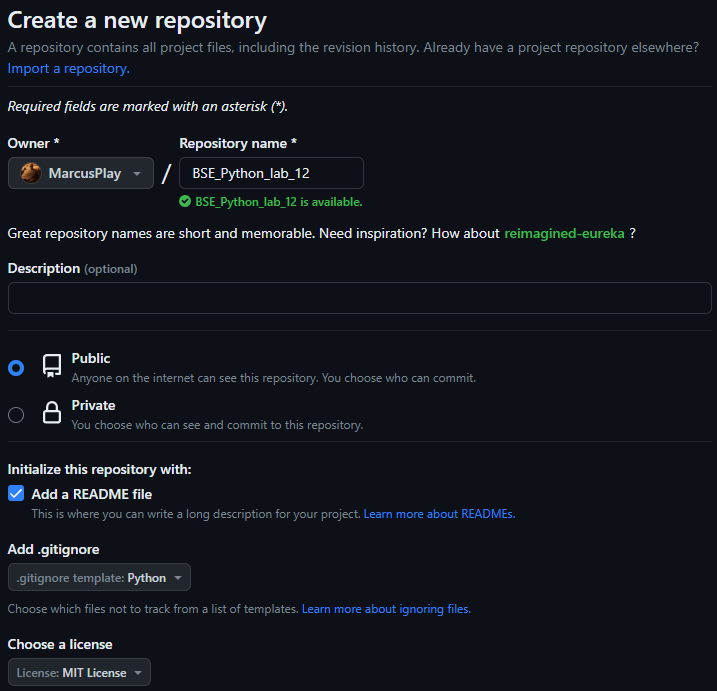
Ставрополь, 2023 г.

**Тема:** Рекурсия в языке Python

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

**Методика и порядок выполнения работы**

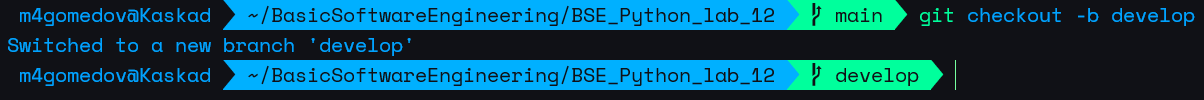
1. Изучить теоретический материал работы.
2. Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT и язык программирования Python.



1. Выполните клонирование созданного репозитория.



1. Дополните файл .gitignore необходимыми правилами для работы с IDE PyCharm.
2. Организуйте свой репозиторий в соответствие с моделью ветвления git-flow.



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 измениться скорость работы рекурсивных версий функций factorial и fib при использовании декоратора lru\_cache? Приведите в отчет и обоснуйте полученные результаты.

*import* timeit

*from* functools *import* lru\_cache

*import* sys

sys.set\_int\_max\_str\_digits(0)

sys.setrecursionlimit(10000)

*# Итеративная версия факториала*

def factorial\_iterative(n):

    result = 1

*for* i *in* range(1, n + 1):

        result \*= i

*return* result

*# Рекурсивная версия факториала*

def factorial\_recursive(n):

*if* n == 0 or n == 1:

*return* 1

*return* n \* factorial\_recursive(n - 1)

*# Итеративная версия чисел Фибоначчи*

def fib\_iterative(n):

    a, b = 0, 1

*for* \_ *in* range(n):

        a, b = b, a + b

*return* a

*# Рекурсивная версия чисел Фибоначчи*

def fib\_recursive(n):

*if* n == 0:

*return* 0

*elif* n == 1:

*return* 1

*else*:

*return* fib\_recursive(n - 1) + fib\_recursive(n - 2)

*# Декоратор lru\_cache для рекурсивных функций*

@lru\_cache(maxsize=None)

def factorial\_recursive\_cached(n):

*if* n == 0 or n == 1:

*return* 1

*return* n \* factorial\_recursive\_cached(n - 1)

@lru\_cache(maxsize=None)

def fib\_recursive\_cached(n):

*if* n == 0:

*return* 0

*elif* n == 1:

*return* 1

*else*:

*return* fib\_recursive\_cached(n - 1) + fib\_recursive\_cached(n - 2)

def main(x, y):

    print(f"\nfactorial\_iterative({x}) --------> {timeit.timeit(str(factorial\_iterative(x)), number=10000)}")

    print(f"factorial\_recursive({x}) --------> {timeit.timeit(str(factorial\_recursive(x)), number=10000)}")

    print(f"factorial\_recursive\_cached({x}) -> {timeit.timeit(str(factorial\_recursive\_cached(x)), number=10000)}")

    print(f"\nfib\_iterative({y}) ----------------> {timeit.timeit(str(fib\_iterative(y)), number=10000)}")

    print(f"fib\_recursive({y}) ----------------> {timeit.timeit(str(fib\_recursive(y)), number=10000)}")

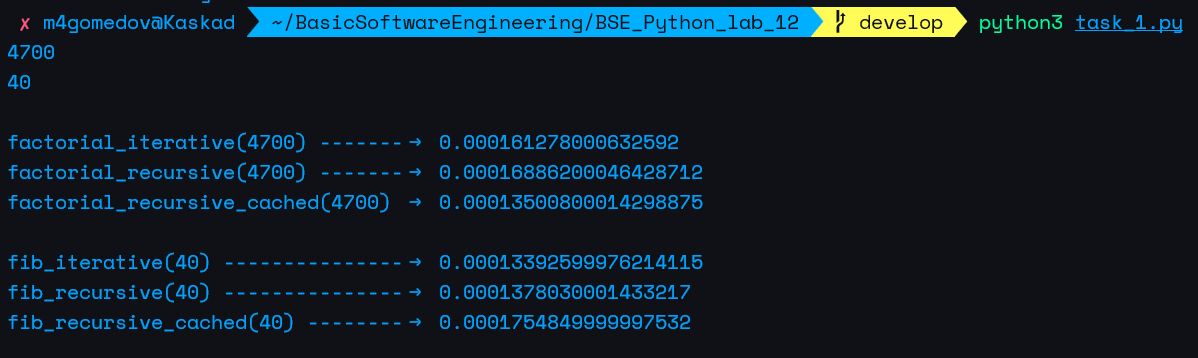
    print(f"fib\_recursive\_cached({y}) ---------> {timeit.timeit(str(fib\_recursive\_cached(y)), number=10000)}\n")

*if* \_\_name\_\_=="\_\_main\_\_":

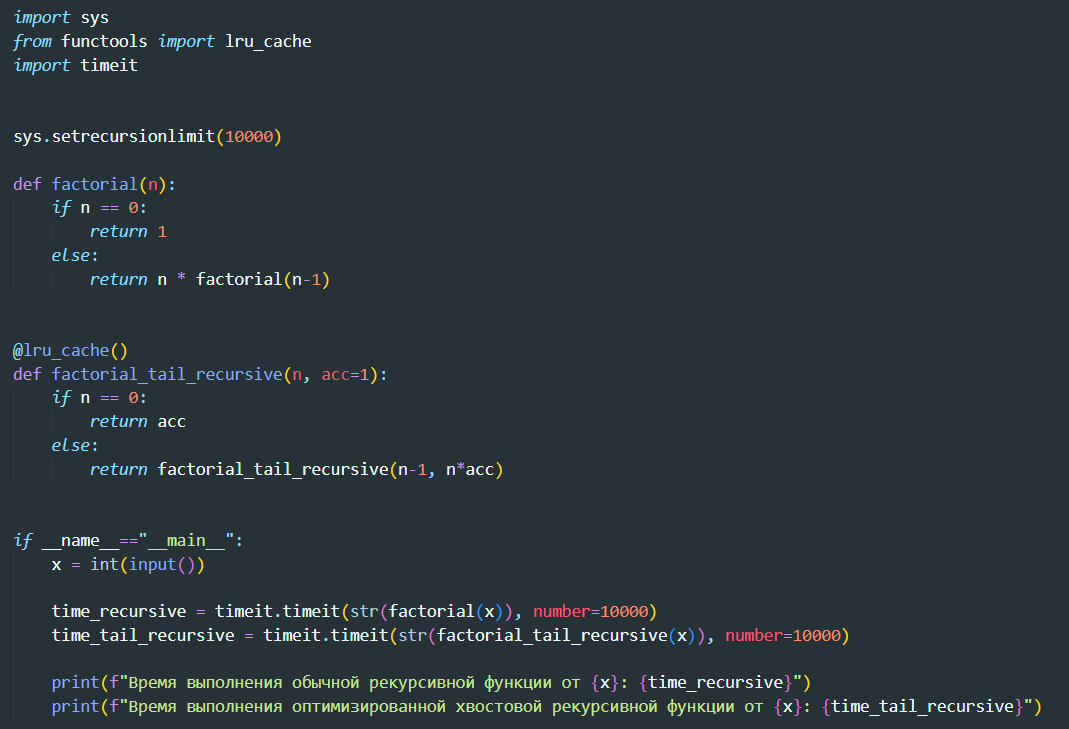
    x = int(input())

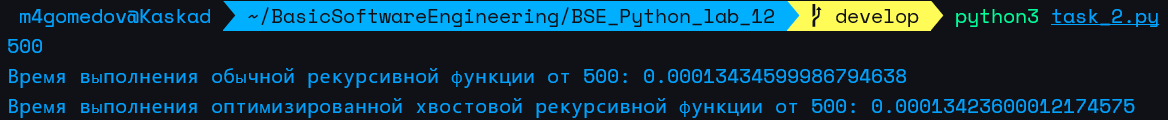
    y = int(input())

    main(x, y)

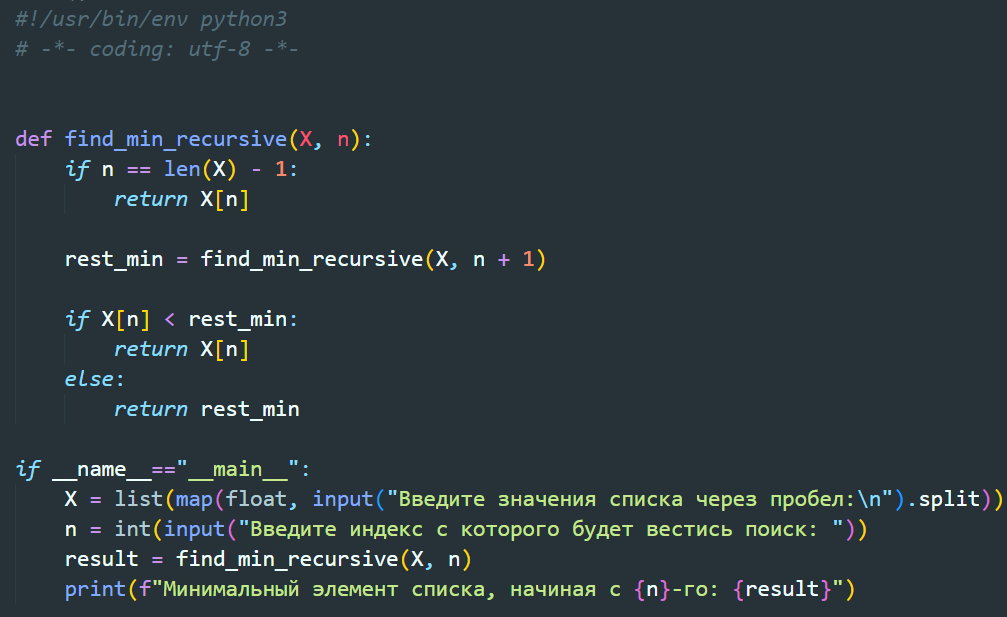


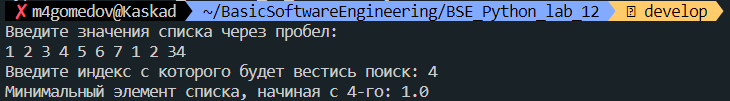
1. Самостоятельно проработайте пример с оптимизацией хвостовых вызовов в Python. С помощью пакета timeit оцените скорость работы функций factorial и fib с использованием интроспекции стека и без использования интроспекции стека. Приведите полученные результаты в отчет.



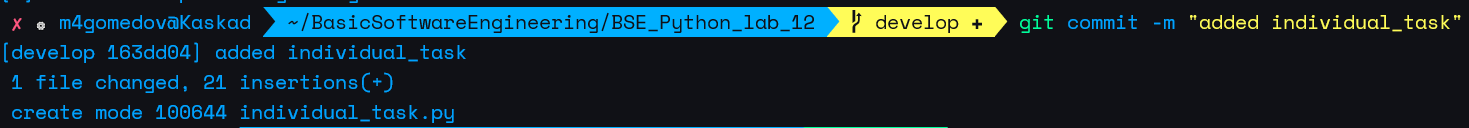


1. Выполните индивидуальные задания. Приведите в отчете скриншоты работы программ решения индивидуального задания.  
   Дан список X из вещественных чисел. Найти минимальный элемент списка, используя вспомогательную рекурсивную функцию, находящую минимум среди последних элементов списка X, начиная с n-гo.

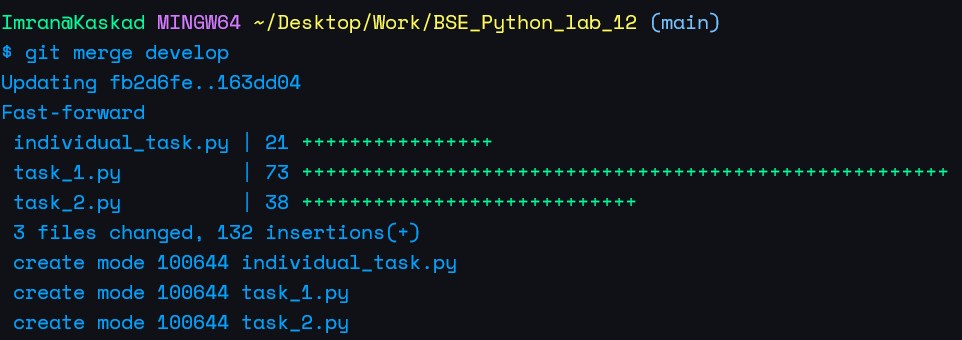




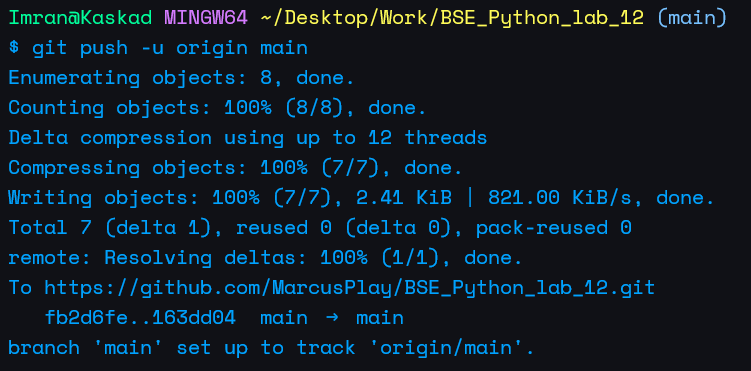
1. Зафиксируйте сделанные изменения в репозитории.



1. Добавьте отчет по лабораторной работе в *формате PDF* в папку *doc* репозитория.
2. Зафиксируйте изменения.
3. Выполните слияние ветки для разработки с веткой *master* / *main*.



1. Отправьте сделанные изменения на сервер GitHub.



1. Отправьте адрес репозитория GitHub на электронный адрес преподавателя.

**Вопросы для защиты работы**

1. Для чего нужна рекурсия?

Рекурсия — это подход в программировании, при котором функция вызывает саму себя. Она применяется для решения задач, которые могут быть разбиты на более простые подзадачи.

2. Что называется базой рекурсии?

База рекурсии — это условие, при котором функция перестает вызывать саму себя и возвращает конкретное значение. Это условие необходимо, чтобы избежать бесконечного цикла вызовов.

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

Стек программы представляет собой структуру данных, используемую для хранения информации о вызовах функций в программе. При вызове функции информация о текущем состоянии помещается в вершину стека, а при завершении функции эта информация удаляется.

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

Для получения текущего значения максимальной глубины рекурсии в Python можно использовать модуль sys и атрибут getrecursionlimit().

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

Если число рекурсивных вызовов превысит максимальную глубину рекурсии в Python, произойдет ошибка "RecursionError: maximum recursion depth exceeded".

6. Как изменить максимальную глубину рекурсии в языке Python?

Максимальную глубину рекурсии в Python можно изменить с помощью функции sys.setrecursionlimit(). Однако, изменение этого значения требует осторожности, так как слишком большое значение может привести к проблемам с памятью или крашам программы.

7. Каково назначение декоратора lru\_cache?

Декоратор lru\_cache используется для кеширования результатов вызова функции с определенными аргументами. Он помогает избежать повторных вычислений, сохраняя результаты в памяти.

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

Хвостовая рекурсия — это случай, когда рекурсивный вызов является последней операцией в функции. В некоторых языках программирования, таких как Scheme, хвостовая рекурсия может быть оптимизирована компилятором для уменьшения использования стека. В Python оптимизация хвостовых вызовов не поддерживается стандартным интерпретатором CPython, но она может быть полезной в других языках.