

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ
ФЕДЕРАЦИИ**

**Федеральное государственное автономное
образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

Кафедра инфокоммуникаций

**Отчет по лабораторной работе № 2.9
по дисциплине «Основы программной инженерии»**

Выполнил студент группы
ПИЖ-б-о-21-1

Трушева В. О. .« » 2022г.

Подпись студента _____

Работа защищена «
» _____ 20__ г.

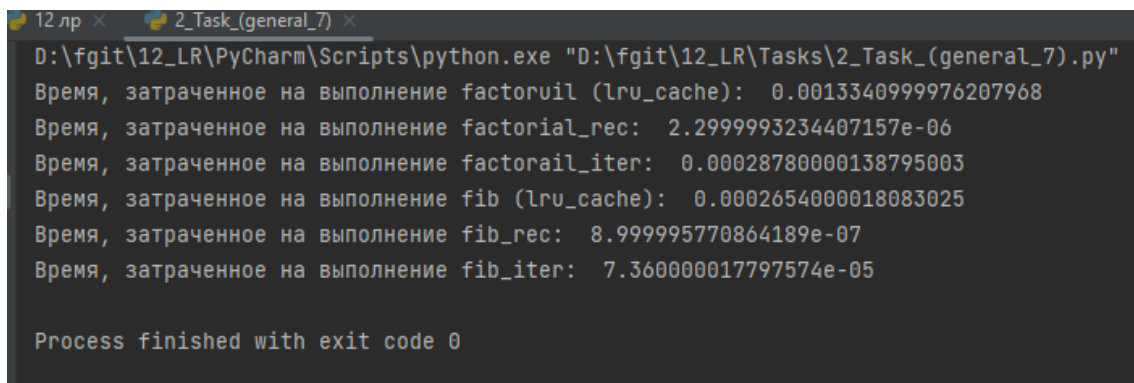
Проверила Воронкин Р.А.

(подпись)

Ставрополь 2022

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.3. Выполните клонирование созданного репозитория.
4. Дополните файл .gitignore необходимыми правилами для работы с IDE PyCharm.
5. Организуйте свой репозиторий в соответствие с моделью ветвления git-flow.
6. Создайте проект PyCharm в папке репозитория.
7. 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.

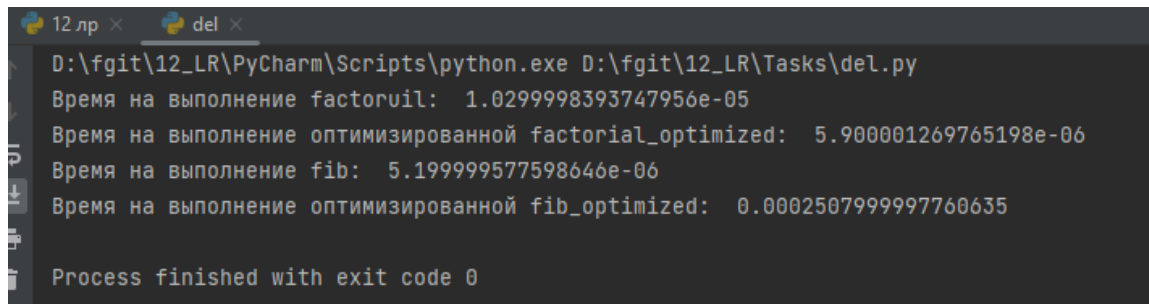


```
12 лр x 2_Task_(general_7) x
D:\fgit\12_LR\PyCharm\Scripts\python.exe "D:\fgit\12_LR\Tasks\2_Task_(general_7).py"
Время, затраченное на выполнение factoruil (lru_cache): 0.0013340999976207968
Время, затраченное на выполнение factorial_rec: 2.2999993234407157e-06
Время, затраченное на выполнение factorail_iter: 0.00028780000138795003
Время, затраченное на выполнение fib (lru_cache): 0.0002654000018083025
Время, затраченное на выполнение fib_rec: 8.999995770864189e-07
Время, затраченное на выполнение fib_iter: 7.360000017797574e-05

Process finished with exit code 0
```

Рисунок 1 – Результат выполнения программы

8. 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.



```
D:\fgit\12_LR\PyCharm\Scripts\python.exe D:\fgit\12_LR\Tasks\del.py
Время на выполнение factoruil: 1.0299998393747956e-05
Время на выполнение оптимизированной factorial_optimized: 5.900001269765198e-06
Время на выполнение fib: 5.199999577598646e-06
Время на выполнение оптимизированной fib_optimized: 0.0002507999997760635

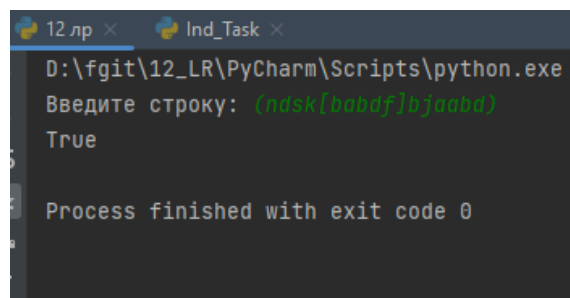
Process finished with exit code 0
```

Рисунок 2 – Результат выполнения программы

9. Выполните индивидуальные задания. Приведите в отчете скриншоты работы программ решения индивидуального задания.

Вариант – 2. В строке могут присутствовать скобки как круглые, так и квадратные скобки. Каждой открывающей скобке соответствует закрывающая того же типа (круглой – круглая квадратной- квадратная). Напишите рекурсивную функцию, проверяющую правильность расстановки скобок в этом случае.

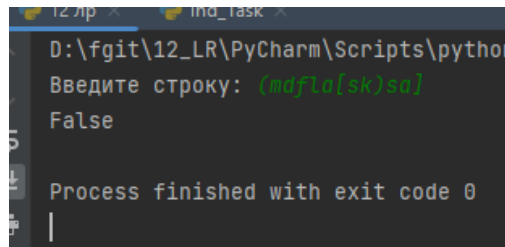
Пример неправильной расстановки: ([)].



```
D:\fgit\12_LR\PyCharm\Scripts\python.exe
Введите строку: (nask[banaf]njoaba)
True

Process finished with exit code 0
```

Рисунок 3 – Результат выполнения программы



```
D:\fgit\12_LR\PyCharm\Scripts\python
Введите строку: (mafla{sk}sa)
False
Process finished with exit code 0
```

Рисунок 4 – Результат выполнения программы

10. Зафиксируйте сделанные изменения в репозитории.

11. Добавьте отчет по лабораторной работе в формате PDF в папку doc репозитория.

Зафиксируйте изменения.

12. Выполните слияние ветки для разработки с веткой master / main.

13. Отправьте сделанные изменения на сервер GitHub.

14. Отправьте адрес репозитория GitHub на электронный адрес преподавателя.

Контрольные вопросы:

1. Для чего нужна рекурсия?

В программировании рекурсия — вызов функции (процедуры) из неё же самой, непосредственно (простая рекурсия) или через другие функции (сложная или косвенная рекурсия). Рекурсивная программа позволяет описать повторяющееся или даже потенциально бесконечное вычисление, причём без явных повторений частей программы и использования циклов.

2. Что называется базой рекурсии?

База рекурсии — это такие аргументы функции, которые делают задачу настолько простой, что решение не требует дальнейших вложенных вызовов.

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

Стек — это структура данных, в которой элементы хранятся в порядке поступления. Стек хранит последовательность данных. Связаны данные так: каждый элемент указывает на тот, который нужно использовать следующим. Это линейная связь — данные идут друг за другом и нужно брать их по очереди. Из середины стека брать нельзя. Главный принцип работы стека — данные, которые попали в стек недавно, используются первыми. Чем раньше попал — тем позже используется. После использования элемент стека исчезает, и верхним становится следующий элемент.

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

Функция `sys.getrecursionlimit()` возвращает текущее значение предела рекурсии, максимальную глубину стека интерпретатора Python. Этот предел предотвращает бесконечную рекурсию от переполнения стека языка C и сбоя Python. Это значение может быть установлено с помощью `sys`.

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 RunTime.

6. Как изменить максимальную глубину рекурсии в языке Python?

С помощью `sys.setrecursionlimit(число)`.

7. Каково назначение декоратора `lru_cache`?

Функция `lru_cache` предназначена для мемоизации (предотвращения повторных вычислений), т. е. кэширует результат в памяти. Полезный инструмент, который уменьшает количество лишних вычислений.

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. Подобный вид рекурсии примечателен тем, что может быть легко заменён на итерацию путём формальной и гарантированно корректной перестройки кода функции. Оптимизация хвостовой рекурсии путём преобразования ее в плоскую итерацию реализована во многих оптимизирующих компиляторах. В некоторых функциональных языках программирования спецификация гарантирует обязательную оптимизацию хвостовой рекурсии.

Типовой механизм реализации вызова функции основан на сохранении адреса возврата, параметров и локальных переменных функции в стеке и выглядит следующим образом:

1. В точке вызова в стек помещаются параметры, передаваемые функции, и адрес возврата.

2. Вызываемая функция в ходе работы размещает в стеке собственные локальные переменные.

3. По завершении вычислений функция очищает стек от своих локальных переменных, записывает результат (обычно — в один из регистров процессора).

4. Команда возврата из функции считывает из стека адрес возврата и выполняет переход по этому адресу. Либо непосредственно перед, либо сразу после возврата из функции стек очищается от параметров.