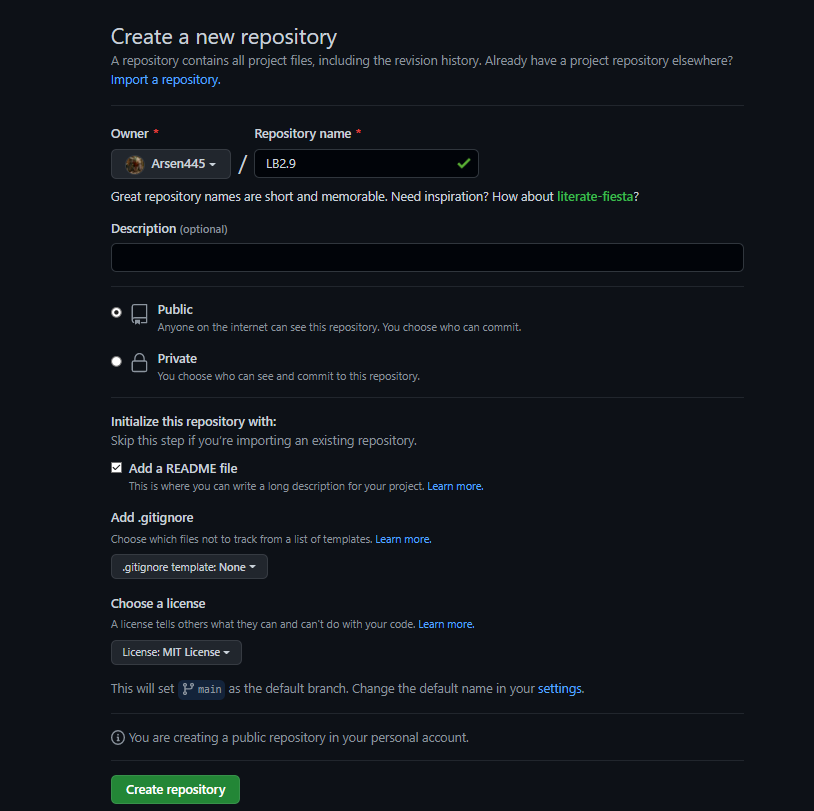
Лабораторная работа №4

Выполнил Эсеналиев Арсен

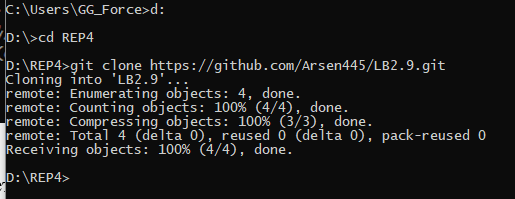
ИВТ-б-о-21-1

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

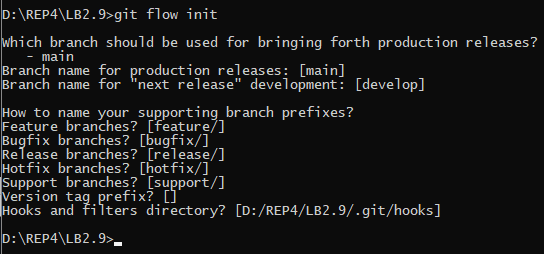
1. Создал общедоступный репозиторий на GitHub c MIT



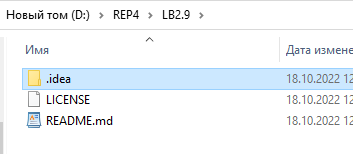
1. Выполнил клонирование созданного репозитория.



3. Организовал свой репозиторий в соответствие с моделью ветвления git-flow. (Перешел с главной main на develop)



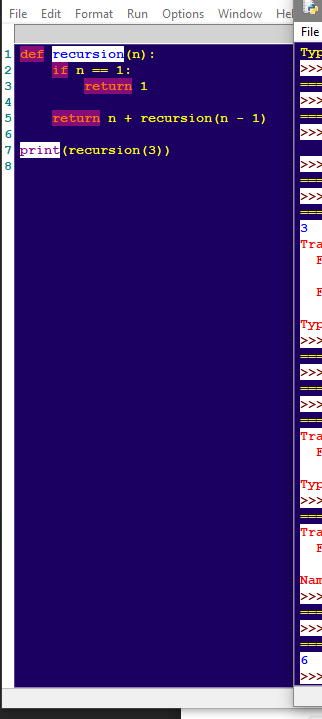
1. Создал проект PyCharm в папке репозитория.



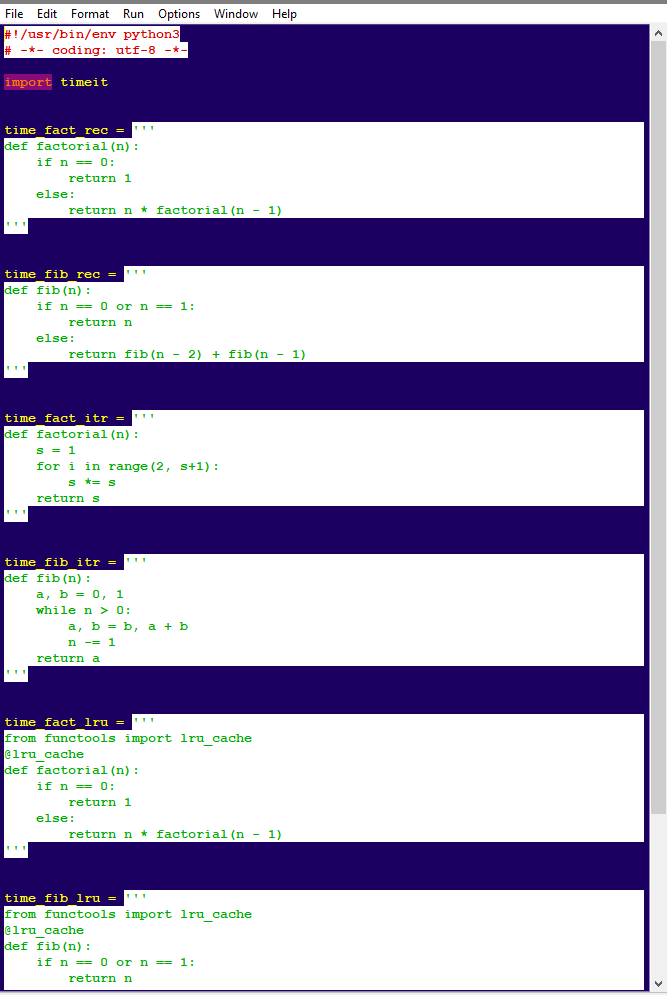
1. Дополнил файл .gitignore необходимыми правилами для работы с IDE PyCharm.



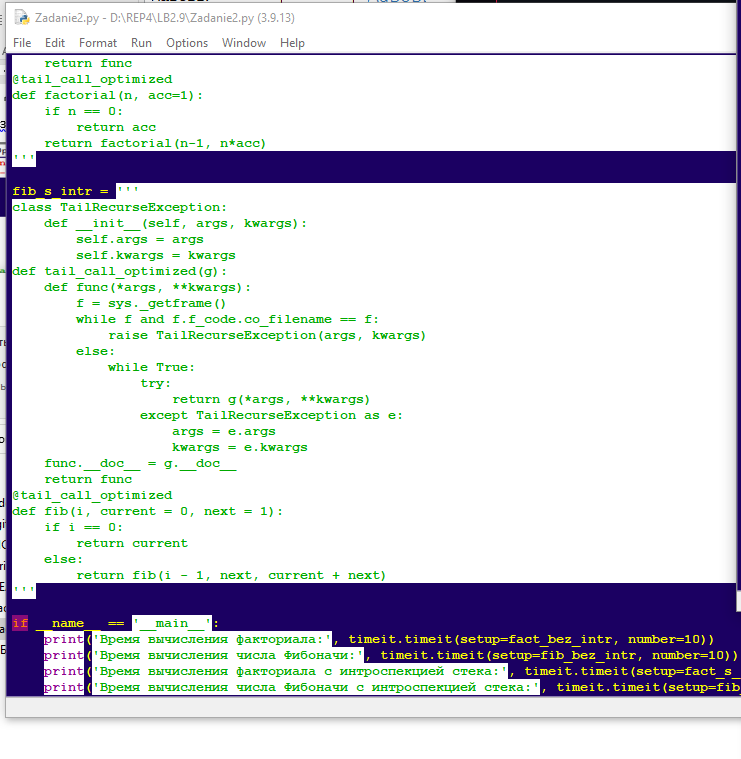
1. Проработал пример лабораторной работы.

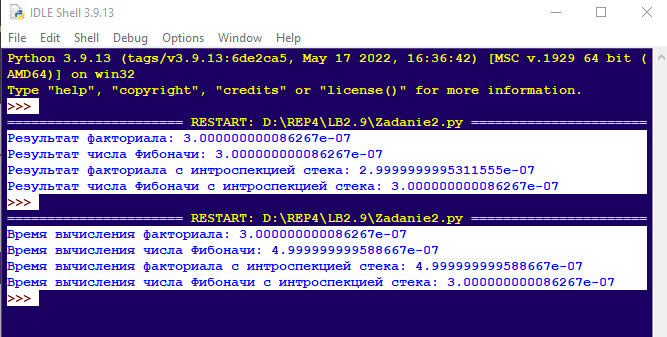


1. Самостоятельно изучите работу со стандартным пакетом Python timeit . Оцените с помощью этого модуля скорость работы итеративной и рекурсивной версий функций factorial и fib . Во сколько раз измениться скорость работы рекурсивных версий функций factorial и fib при использовании декоратора lru\_cache ? Приведите в отчет и обоснуйте полученные результаты..

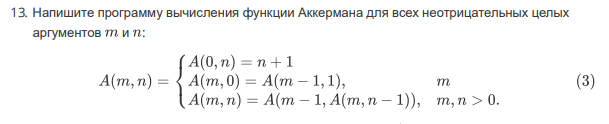


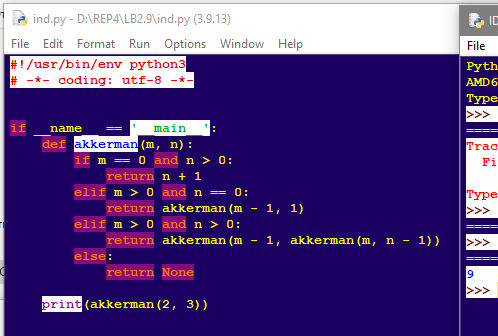
1. Самостоятельно проработайте пример с оптимизацией хвостовых вызовов в Python. С помощью пакета timeit оцените скорость работы функций factorial и fib с использованием интроспекции стека и без использования интроспекции стека. Приведите полученные результаты в отчет.



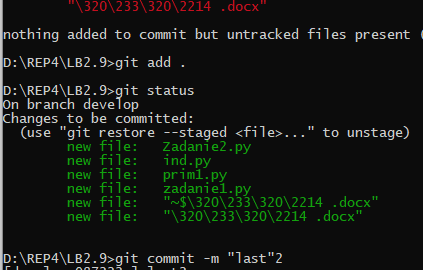
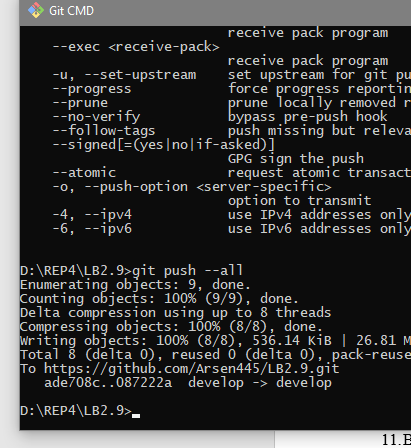


9. Индивидуальное задание

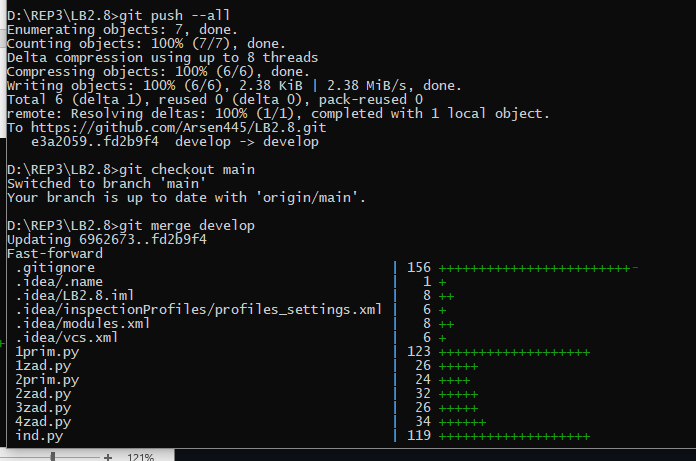




1. Зафиксируйте сделанные изменения в репозитории.



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



**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 предназначается 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 её 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 поэтому адресу. Либо непосредственно перед, либо сразу после возврата из функции стек очищается от параметров.