

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ  
ФЕДЕРАЦИИ**  
**Федеральное государственное автономное  
образовательное учреждение высшего образования  
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

**Кафедра инфокоммуникаций**

**«Основы ветвления Git»**

**Отчет по лабораторной работе № 2.9**

**по дисциплине «Основы программной инженерии»**

Выполнил студент группы ПИЖ-б-о-21-1

Образцова М. Д.. «17» декабря 2022г.

Подпись студента \_\_\_\_\_

Работа защищена « » \_\_\_\_\_ 20\_\_ г.

Проверил Воронкин Р.А. \_\_\_\_\_

(подпись)

Ставрополь 2022


## МЕТОДИКА И ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1. Создала репозиторий в GitHub «OPI\_LR\_12» в который добавила .gitignore для работы с IDE PyCharm с ЯП Python, выбрала лицензию MIT.

---

Owner \*

Repository name \*

 obrMaria ▾


/

OPI\_LR\_12 ✓


Great repository names are short and memorable. Need inspiration? How about [fuzzy-carnival?](#)

Description (optional)

---

☒  **Public**

Anyone on the internet can see this repository. You choose who can commit.

☐  **Private**

You choose who can see and commit to this repository.

---

**Initialize this repository with:**

Skip this step if you're importing an existing repository.

☒ **Add a README file**

This is where you can write a long description for your project. [Learn more.](#)

**Add .gitignore**


Choose which files not to track from a list of templates. [Learn more.](#)

.gitignore template: Python ▾


**Choose a license**

A license tells others what they can and can't do with your code. [Learn more.](#)

License: MIT License ▾

This will set  **main** as the default branch. Change the default name in your [settings](#).

---

 You are creating a public repository in your personal account.

---

Create repository

Рисунок 1 – Создание репозитория

```
M@DESKTOP-UVM9NOL MINGW64 ~/desktop/OPI_LR_12 (main)
$ git flow init

Which branch should be used for bringing forth production releases?
- main
Branch name for production releases: [main]
Branch name for "next release" development: [develop]

How to name your supporting branch prefixes?
Feature branches? [feature/]
Bugfix branches? [bugfix/]
Release branches? [release/]
Hotfix branches? [hotfix/]
Support branches? [support/]
Version tag prefix? []
Hooks and filters directory? [C:/Users/M/desktop/OPI_LR_12/.git/hooks]

M@DESKTOP-UVM9NOL MINGW64 ~/desktop/OPI_LR_12 (develop)
```

Рисунок 2 – Организация репозитория в соответствии с моделью ветвления git-flow

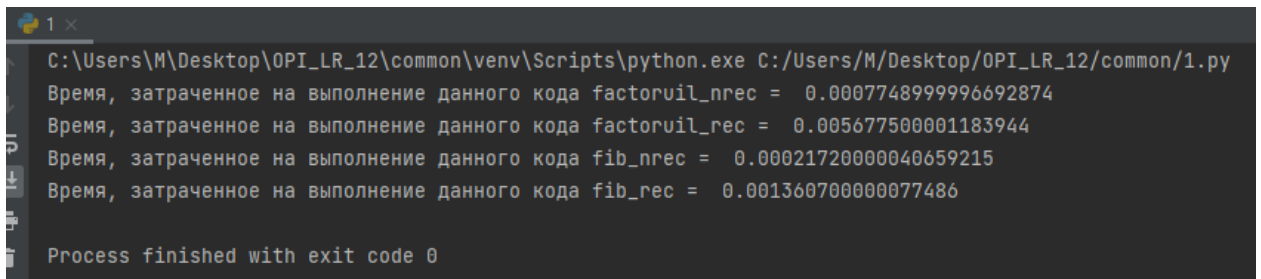
```
M@DESKTOP-UVM9NOL MINGW64 ~/desktop/OPI_LR_12 (develop)
$ git add .

M@DESKTOP-UVM9NOL MINGW64 ~/desktop/OPI_LR_12 (develop)
$ git commit -m "gitignore"
[develop 35c144c] gitignore
1 file changed, 140 insertions(+), 2 deletions(-)

M@DESKTOP-UVM9NOL MINGW64 ~/desktop/OPI_LR_12 (develop)
$
```

Рисунок 3 – Изменение .gitignore

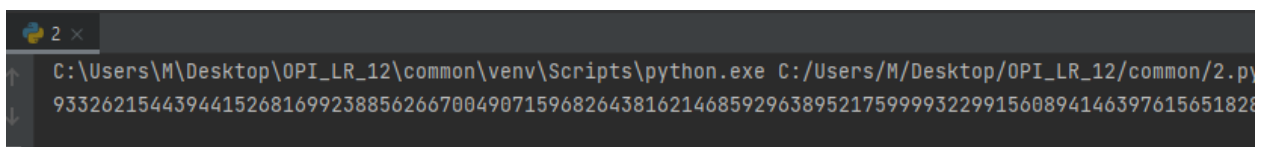
7. Самостоятельно изучите работу со стандартным пакетом Python `timeit`. Оцените с помощью этого модуля скорость работы итеративной и рекурсивной версий функций `factorial` и `fib`. Во сколько раз изменится скорость работы рекурсивных версий функций `factorial` и `fib` при использовании декоратора `lru_cache`?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



```
C:\Users\M\Desktop\OPI_LR_12\common\venv\Scripts\python.exe C:/Users/M/Desktop/OPI_LR_12/common/1.py
Время, затраченное на выполнение данного кода factoruil_nrec = 0.0007748999996692874
Время, затраченное на выполнение данного кода factoruil_rec = 0.005677500001183944
Время, затраченное на выполнение данного кода fib_nrec = 0.00021720000040659215
Время, затраченное на выполнение данного кода fib_rec = 0.001360700000077486

Process finished with exit code 0
```

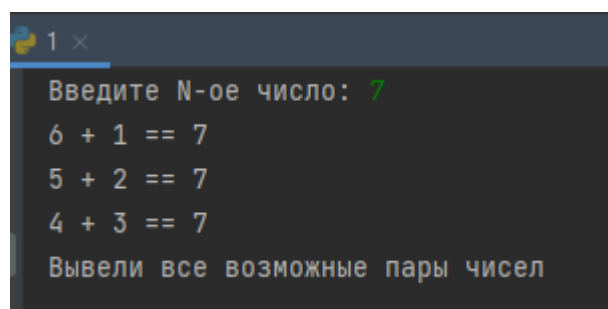
8. Самостоятельно проработайте пример с оптимизацией хвостовых вызовов в Python. С помощью пакета `timeit` оцените скорость работы функций `factorial` и `fib` с использованием интроспекции стека и без использования интроспекции стека. Приведите полученные результаты в отчет.



```
C:\Users\M\Desktop\OPI_LR_12\common\venv\Scripts\python.exe C:/Users/M/Desktop/OPI_LR_12/common/2.py
9332621544394415268169923885626670049071596826438162146859296389521759999322991560894146397615651828
```

Индивидуальные задания.

Создайте процедуру, печатающую все возможные представления натурального числа в виде суммы других натуральных чисел.



```
1 x
Введите N-ое число: 7
6 + 1 == 7
5 + 2 == 7
4 + 3 == 7
Вывели все возможные пары чисел
```

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 `RuntimeError`.

**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 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 по этому адресу. Либо непосредственно перед, либо сразу после возврата из функции стек очищается от параметров.