

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

Кафедра инфокоммуникаций

Основы кроссплатформенного программирования

Отчет по лабораторной работе №2.9

Тема: «Рекурсия в языке Python»

Выполнил студент группы

ИВТ-б-о-21-1

Богадуров В.И. « » _____ 20__ г.

Подпись студента _____

Работа защищена « » _____ 20__ г.

Проверил доцент

Кафедры инфокоммуникаций, старший
преподаватель

Воронкин Р.А.

(подпись)

Ставрополь 2022

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

Ход работы:

1. Создал репозиторий в GitHub, дополнил правила в .gitignore для работы с IDE PyCharm с ЯП Python, выбрал лицензию MIT, клонировал его на компьютер и организовал в соответствии с моделью ветвления git-flow.

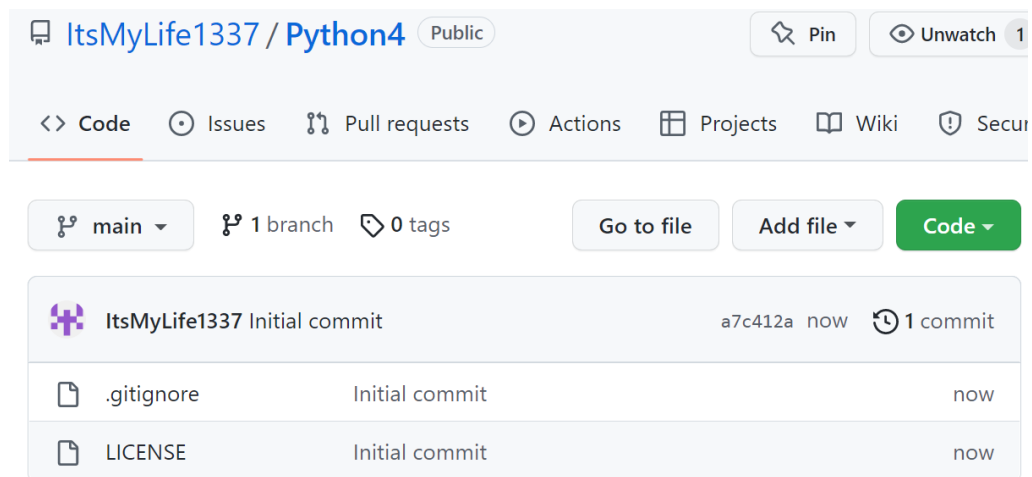


Рисунок 1.1 – Созданный репозиторий

```
.gitignore – Блокнот
Файл Правка Формат Вид Справка
.idea/
# Created by https://www.toptal.com/developers/gitignore/api/python,pycharm
# Edit at https://www.toptal.com/developers/gitignore?templates=python,pycharm

### PyCharm ###
# Covers JetBrains IDEs: IntelliJ, RubyMine, PhpStorm, AppCode, PyCharm, CLion, A
# Reference: https://intellij-support.jetbrains.com/hc/en-us/articles/206544839

# User-specific stuff
.idea/**/workspace.xml
.idea/**/tasks.xml
.idea/**/usage.statistics.xml
.idea/**/dictionaries
.idea/**/shelf

# AWS User-specific
.idea/**/aws.xml

# Generated files
.idea/**/contentModel.xml

# Sensitive or high-churn files
.idea/**/dataSources/
.idea/**/dataSources.ids
.idea/**/dataSources.local.xml
```

Рисунок 1.2 – Дополнил правила в .gitignore

```
c:\Users\Admin\Desktop\git\Python4>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/Admin/Desktop/git/Python4/.git/hooks]

c:\Users\Admin\Desktop\git\Python4>
```

Рисунок 1.3 – Организация репозитория в соответствии с моделью ветвления git-flow

2. Создал проект Pycharm в папке репозитория, проработал примеры ЛР.

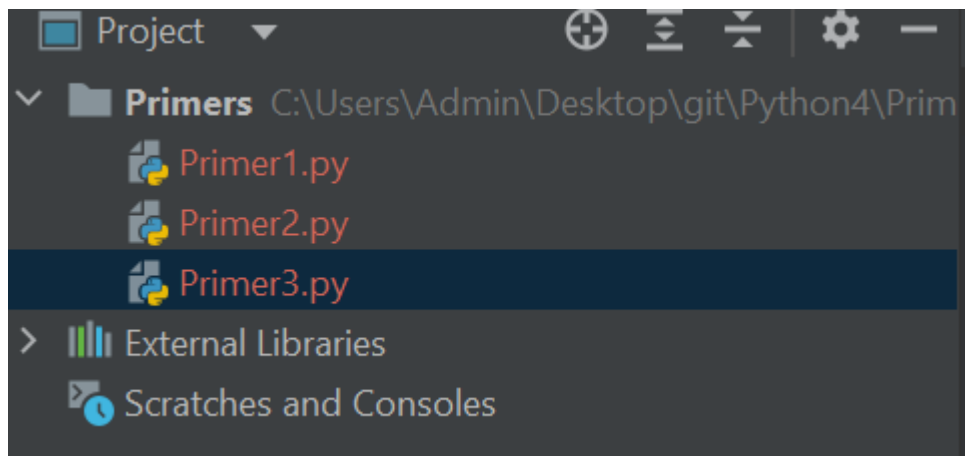


Рисунок 2.1 – Созданные проекты

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

def countdown(n):
    if n == 0:
        print("Blastoff!")
    else:
        print(n)
        countdown(n-1)
```

Primer1 x

C:\Users\Admin\AppData\Local\Programs\Python\Python38\python.exe

Process finished with exit code 0

Рисунок 2.2 – Пример №1

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

def find_max(seq, max_so_far):
    if not seq:
        return max_so_far
    if max_so_far < seq[0]:
        return find_max(seq[1:], seq[0])
    else:
        return find_max(seq[1:], max_so_far)
```

Primer2 x

C:\Users\Admin\AppData\Local\Programs\Python\Python38\python.exe

Process finished with exit code 0

Рисунок 2.3 – Пример №2

```
10
11 class TailRecurseException(Exception):
12     def __init__(self, args, kwargs):
13         self.args = args
14         self.kwargs = kwargs
15
16
17 def tail_call_optimized(g):
18     """
19     Эта функция не работает, если функция деко
20     """
21
22     def func(*args, **kwargs):
23         f = sys._getframe()
24         factorial() > if n == 0
25
26 Run: Primer3 x
27 C:\Users\Admin\AppData\Local\Programs\Python\Python39\python.exe C:\
28 2846259680917054518906413212119868890148051401
29
30 Process finished with exit code 0
```

Рисунок 2.3 – Результат выполнения примера №3

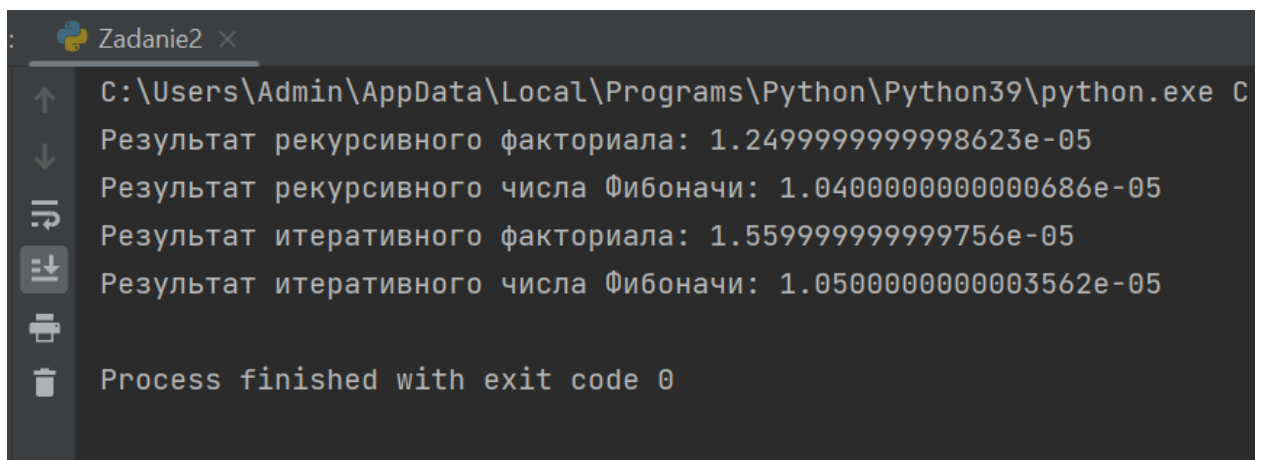
Задание №1. Самостоятельно изучите работу со стандартным пакетом Python timeit. Оцените с помощью этого модуля скорость работы итеративной и рекурсивной версий функций factorial и fib. Во сколько раз измениться скорость работы рекурсивных версий функций factorial и fib при использовании декоратора lru_cache? Приведите в отчет и обоснуйте полученные результаты.

```
Zadanie1 x
C:\Users\Admin\AppData\Local\Programs\Python\Python39\python.exe C:\
Результат рекурсивного факториала: 2.73000000000007874e-05
Результат рекурсивного числа Фибоначи: 2.8000000000000247e-05
Результат итеративного факториала: 2.4099999999999122e-05
Результат итеративного числа Фибоначи: 1.79000000000001248e-05
Результат факториала с декоратором: 1.0499999999996623e-05
Результат числа Фибоначи с декоратором: 1.0399999999993748e-05
Process finished with exit code 0
```

Рисунок 3.1 – Результат выполнения задания №1

Рекурсивное изменить в 2,7 раза, а итеративное в 1,7 раза. Быстрее вычислять декоратором, поскольку функция используется внутри него параллельно меняясь вне его. Декораторы — это, по сути, "обёртки", которые дают нам возможность изменить поведение функции, не изменяя её код.

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.



```
C:\Users\Admin\AppData\Local\Programs\Python\Python39\python.exe C
Результат рекурсивного факториала: 1.2499999999998623e-05
Результат рекурсивного числа Фибоначи: 1.04000000000000686e-05
Результат итеративного факториала: 1.559999999999756e-05
Результат итеративного числа Фибоначи: 1.05000000000003562e-05
Process finished with exit code 0
```

Рисунок 3.2 – Результат выполнения задания №2

Индивидуальное задание. В – 1. Напишите рекурсивную функцию, проверяющую правильность расстановки скобок в строке.

При правильной расстановке выполняются условия:

- количество открывающих и закрывающих скобок равно;
- внутри любой пары открывающая — соответствующая закрывающая скобка, скобки расставлены правильно.

Примеры неправильной расстановки: `)(, ()(, ()())` и т. п.

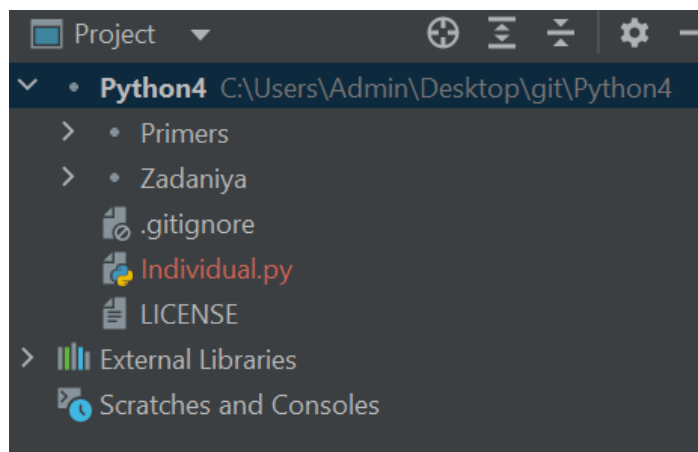


Рисунок 4.1 – Созданный проект

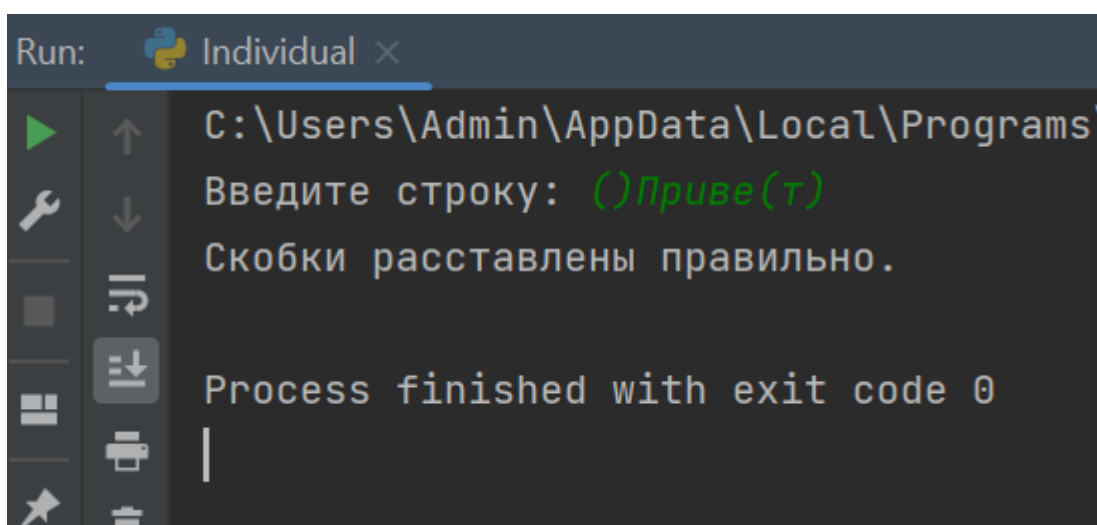


Рисунок 4.2 – Результат выполнения индивидуального задания

Вывод: в результате выполнения лабораторной работы были приобретены практические навыки и теоретические сведения для работы с рекурсивными функциями при написании программ с помощью языка программирования Python версии 3.x.

Ответы на 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.