

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

**Кафедра
инфокоммуникаций
Институт цифрового
развития**

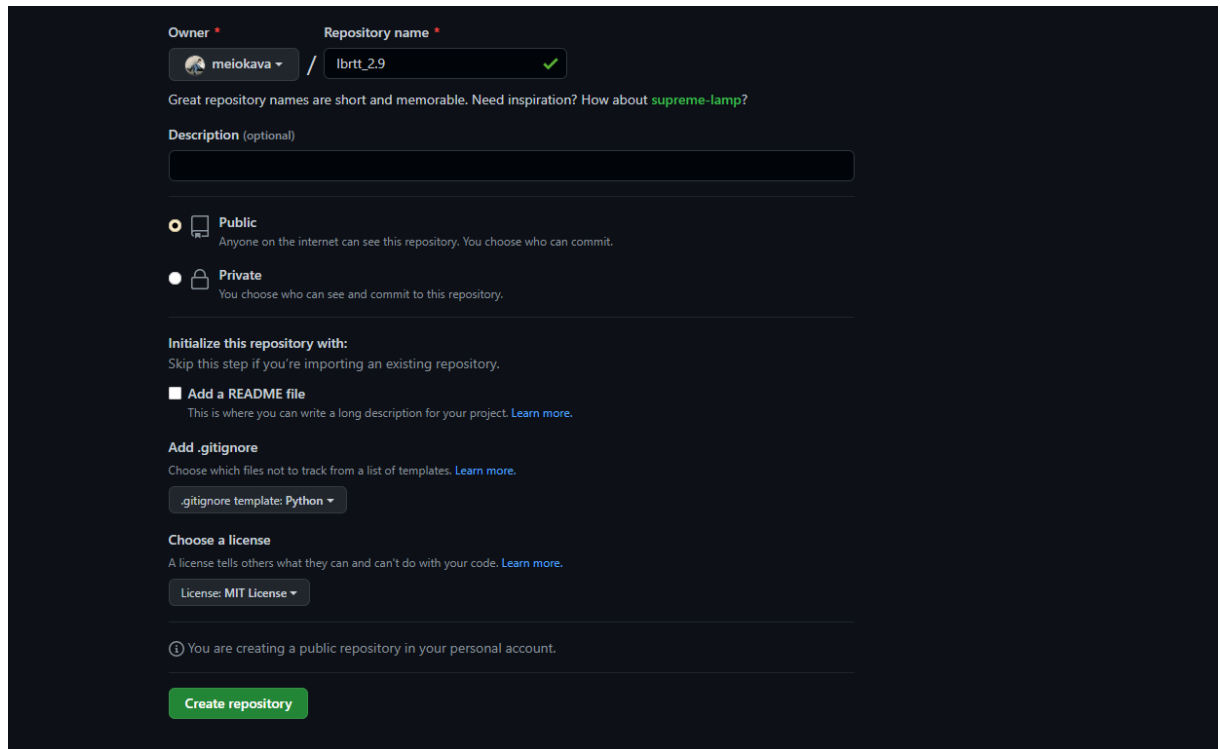
ОТЧЁТ
по лабораторной работе №2.9
Дисциплина: «Основы программной инженерии»
Тема: «Рекурсия в языке Python»

Выполнила:
студентка 2 курса
группы Пиж-б-о-21-1
Джолдошова Мээрим
Бекболотовна

Ставрополь 2022

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

1. Был создан репозиторий в Github в который были добавлены правила gitignore для работы IDE PyCharm, была выбрана лицензия MIT, сам репозиторий был клонирован на локальный сервер и был организован в соответствии с моделью ветвления git-flow.



Owner ^{*} meiokava / Repository name ^{*} lbrtt_2.9 ✓

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

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

① You are creating a public repository in your personal account.

Create repository

Рисунок 1 – Создание репозитория

```
c:\Users\мвидео>cd/d c:\lbrtt_2.9
c:\lbrtt_2.9>git clone https://github.com/meiokava/lbrtt_2.9.git
Cloning into 'lbrtt_2.9'...
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (4/4), done.
c:\lbrtt_2.9>
```

Рисунок 2 – Клонирование репозитория

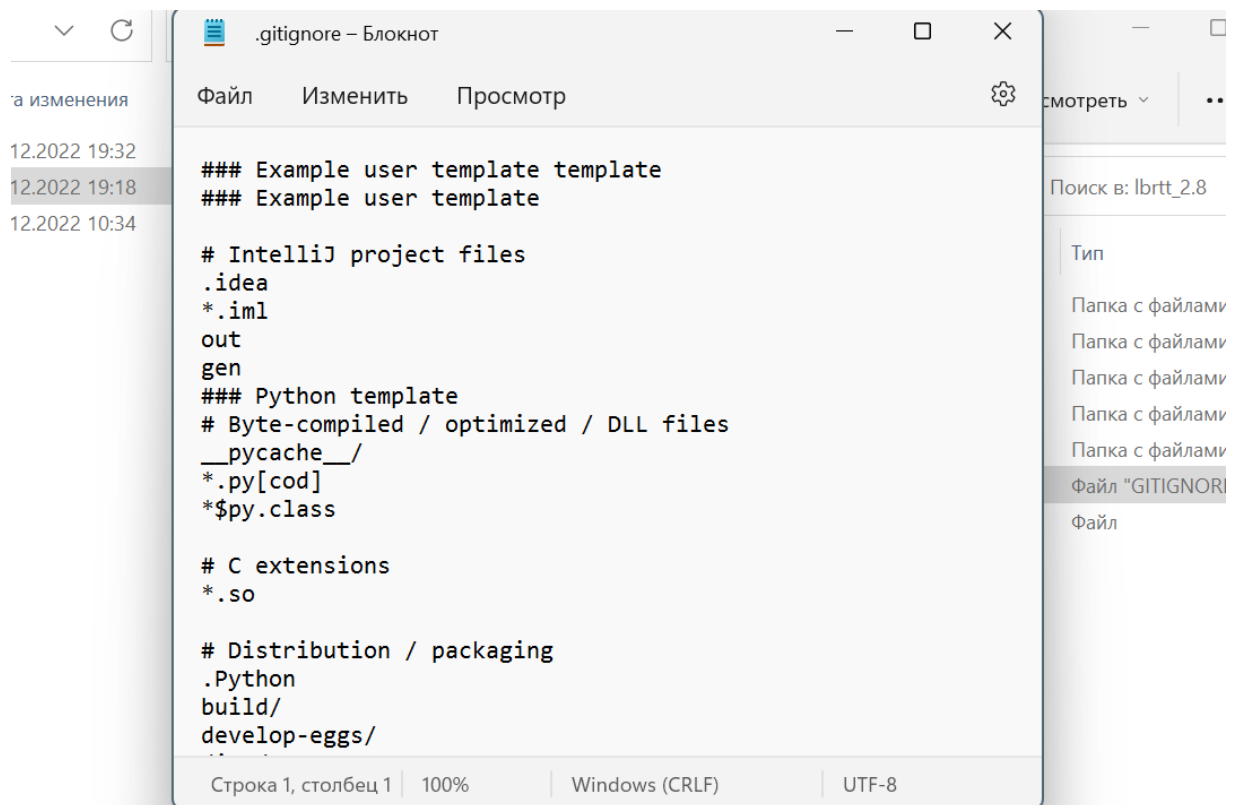


Рисунок 3 – Изменение gitignore

2. Была создана папка PyCharm в которой хранятся примеры из лабораторной работы.

Имя	Дата изменения	Тип	Па
.git	16.12.2022 9:20	Папка с файлами	
PyCharm	16.12.2022 9:20	Папка с файлами	
.gitignore	16.12.2022 9:19	Файл "GITIGNORE"	
LICENSE	16.12.2022 9:19	Файл	

Рисунок 4 – Созданная папка PyCharm



Рисунок 5 – Результат работы 1 примера

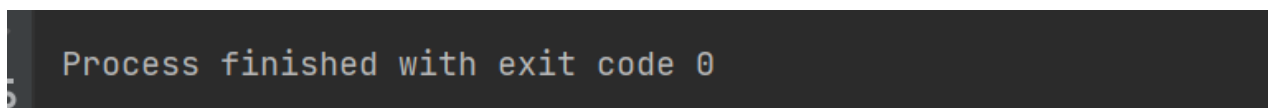


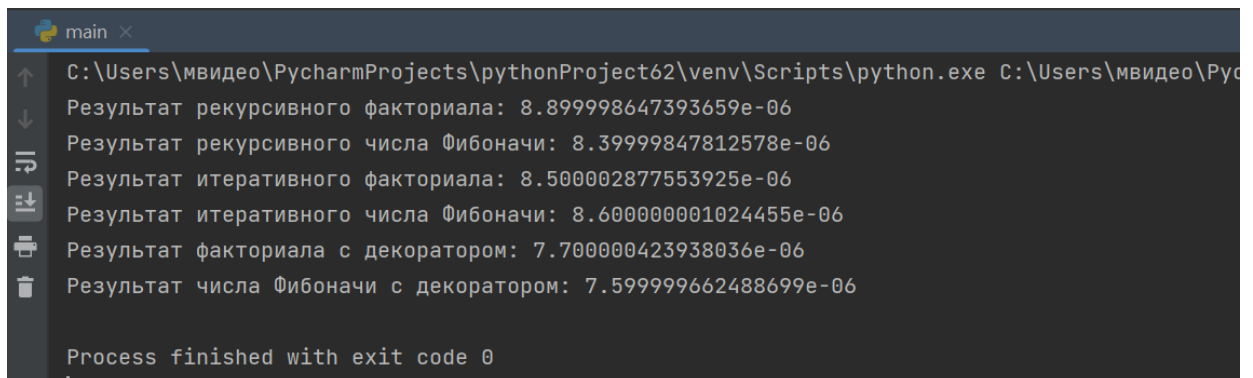
Рисунок 6 – Результат работы 2 примера

```
C:\Users\мвидео\PycharmProjects\pythonProject61\venv\Scripts\python.exe C:\Us
Process finished with exit code 0
```

Рисунок 7 – Результат работы 3 примера

Задание 2:

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.



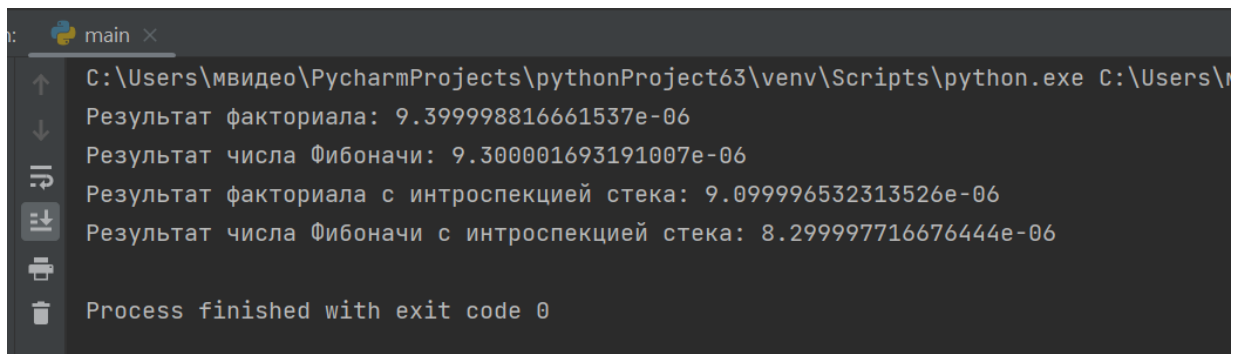
```
main x
C:\Users\мвидео\PycharmProjects\pythonProject62\venv\Scripts\python.exe C:\Users\мвидео\Pyd
Результат рекурсивного факториала: 8.899998647393659e-06
Результат рекурсивного числа Фибоначи: 8.39999847812578e-06
Результат итеративного факториала: 8.500002877553925e-06
Результат итеративного числа Фибоначи: 8.6000000001024455e-06
Результат факториала с декоратором: 7.700000423938036e-06
Результат числа Фибоначи с декоратором: 7.599999662488699e-06
Process finished with exit code 0
```

Рисунок 8 – Получившиеся показатели

Более быстрые вычисления происходят с использованием декоратора (это функция, которая принимает другую функцию в качестве аргумента, модифицирует или улучшает принятую функцию в последствии выдает измененную)

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.



```
main x
C:\Users\мвидео\PycharmProjects\pythonProject63\venv\Scripts\python.exe C:\Users\мвидео\PycharmProjects\pythonProject63\main.py
Результат факториала: 9.399998816661537e-06
Результат числа Фибоначи: 9.300001693191007e-06
Результат факториала с интроспекцией стека: 9.099996532313526e-06
Результат числа Фибоначи с интроспекцией стека: 8.299997716676444e-06
Process finished with exit code 0
```

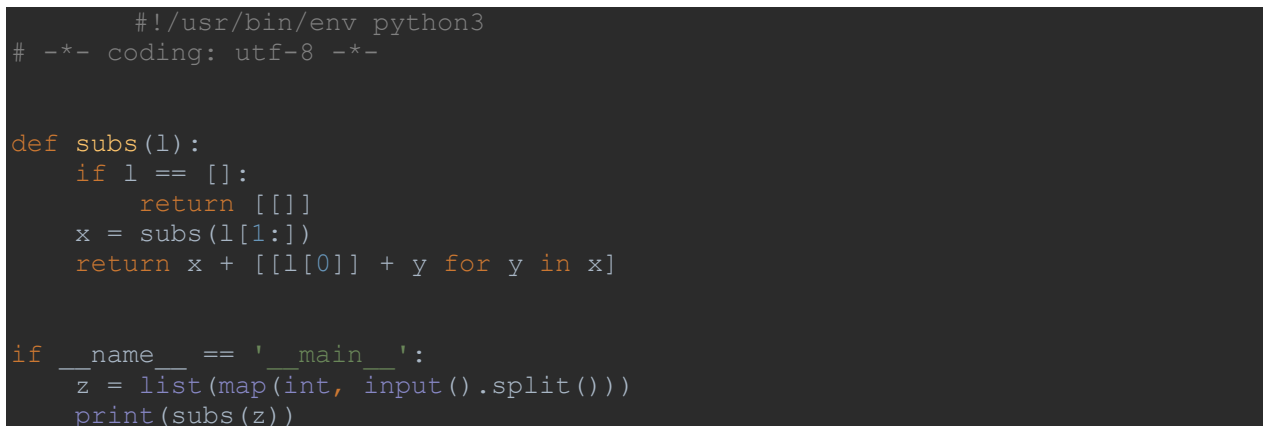
Рисунок 9 – Получившиеся показатели

Использование интроспекции стека сделала процесс вычисления более быстрым (за счет способность объекта во время выполнения получить информацию о его внутренней структуре.)

Индивидуальное задание

Вариант 5

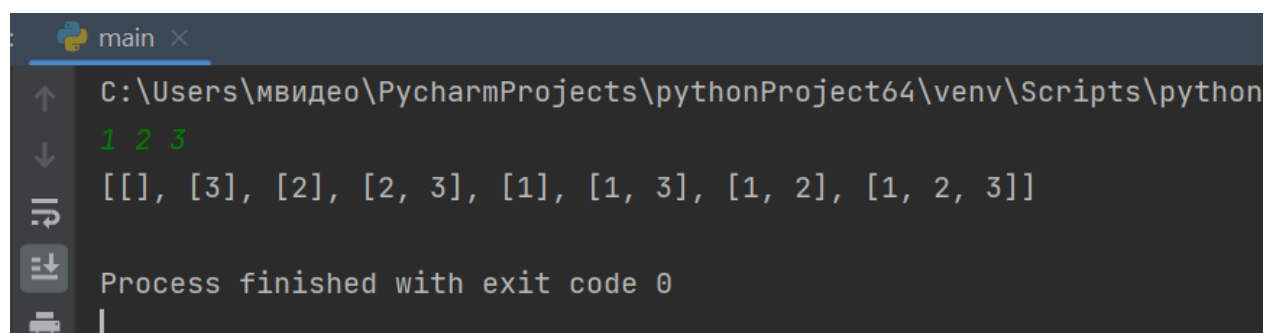
Создайте рекурсивную функцию, печатающую все подмножества множества.



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

def subs(l):
    if l == []:
        return [[]]
    x = subs(l[1:])
    return x + [[l[0]] + y for y in x]

if __name__ == '__main__':
    z = list(map(int, input().split()))
    print(subs(z))
```



```
main x
C:\Users\мвидео\PycharmProjects\pythonProject64\venv\Scripts\python.exe C:\Users\мвидео\PycharmProjects\pythonProject64\main.py
1 2 3
[[], [3], [2], [2, 3], [1], [1, 3], [1, 2], [1, 2, 3]]
Process finished with exit code 0
```

Рисунок 10 – Результат работы программы

```

c:\lbrtt_2.9\lbrtt_2.9>git add .

c:\lbrtt_2.9\lbrtt_2.9>git commit -m "add code"
[develop b5f3e88] add code
7 files changed, 250 insertions(+), 2 deletions(-)
create mode 100644 PyCharm/examp1.py
create mode 100644 PyCharm/examp2.py
create mode 100644 PyCharm/examp3.py
create mode 100644 indiv/indiv1.py
create mode 100644 tasks/task1.py
create mode 100644 tasks/task2.py

c:\lbrtt_2.9\lbrtt_2.9>git push
fatal: The current branch develop has no upstream branch.
To push the current branch and set the remote as upstream, use

    git push --set-upstream origin develop

To have this happen automatically for branches without a tracking
upstream, see 'push.autoSetupRemote' in 'git help config'.

c:\lbrtt_2.9\lbrtt_2.9>git push --set-upstream origin develop
Enumerating objects: 14, done.
Counting objects: 100% (14/14), done.
Delta compression using up to 8 threads
Compressing objects: 100% (11/11), done.

```

Рисунок 11 – Коммит и пуш изменений

```

c:\lbrtt_2.9\lbrtt_2.9>git merge develop
Updating a5065df..b5f3e88
Fast-forward
 .gitignore      | 24 ++++++++
 PyCharm/examp1.py | 12 +++++
 PyCharm/examp2.py | 11 +++++
 PyCharm/examp3.py | 36 ++++++++
 indiv/indiv1.py  | 14 +++++
 tasks/task1.py   | 77 ++++++++
 tasks/task2.py   | 78 ++++++++
7 files changed, 250 insertions(+), 2 deletions(-)
create mode 100644 PyCharm/examp1.py
create mode 100644 PyCharm/examp2.py
create mode 100644 PyCharm/examp3.py
create mode 100644 indiv/indiv1.py
create mode 100644 tasks/task1.py
create mode 100644 tasks/task2.py

c:\lbrtt_2.9\lbrtt_2.9>git push
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/meiokava/lbrtt_2.9.git
 a5065df..b5f3e88  main -> main

c:\lbrtt_2.9\lbrtt_2.9>_

```

Рисунок 12 – Слияние веток main и develop

Вывод: в результате выполнения лабораторной работы были приобретены навыки по работе с рекурсивными функциями при написании программ с помощью языка программирования Python версии 3.x.

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