

Министерство науки и высшего образования Российской Федерации  
Федеральное государственное автономное образовательное учреждение  
высшего образования  
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт цифрового развития  
Кафедра инфокоммуникаций

**ОТЧЕТ**  
**ПО ЛАБОРАТОРНОЙ РАБОТЕ №12**  
**дисциплины «Основы программной инженерии»**

Выполнил:  
Матвеев Александр Иванович  
2 курс, группа ПИЖ-б-о-22-1,  
09.03.04 «Программная инженерия»,  
направленность (профиль) «Разработка и  
сопровождение программного  
обеспечения», очная форма обучения

\_\_\_\_\_  
(подпись)

Проверил Воронкин Роман Александрович

\_\_\_\_\_  
(подпись)

Отчет защищен с оценкой \_\_\_\_\_ Дата защиты \_\_\_\_\_

Ставрополь, 2023 г.

**Тема:** Рекурсия в языке Python.

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

Ход работы.

1. Создание нового репозитория с лицензией MIT.

**Create a new repository**

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

*Required fields are marked with an asterisk (\*).*

**Owner \*** **Repository name \***

SashkaHacker / laba10

✔ laba10 is available.

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

**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:**

☐ **Add a README file**  
This is where you can write a long description for your project. [Learn more about READMEs.](#)

**Add .gitignore**

.gitignore template: Python

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

**Choose a license**

License: MIT License

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

You are creating a public repository in your personal account.

**Create repository**

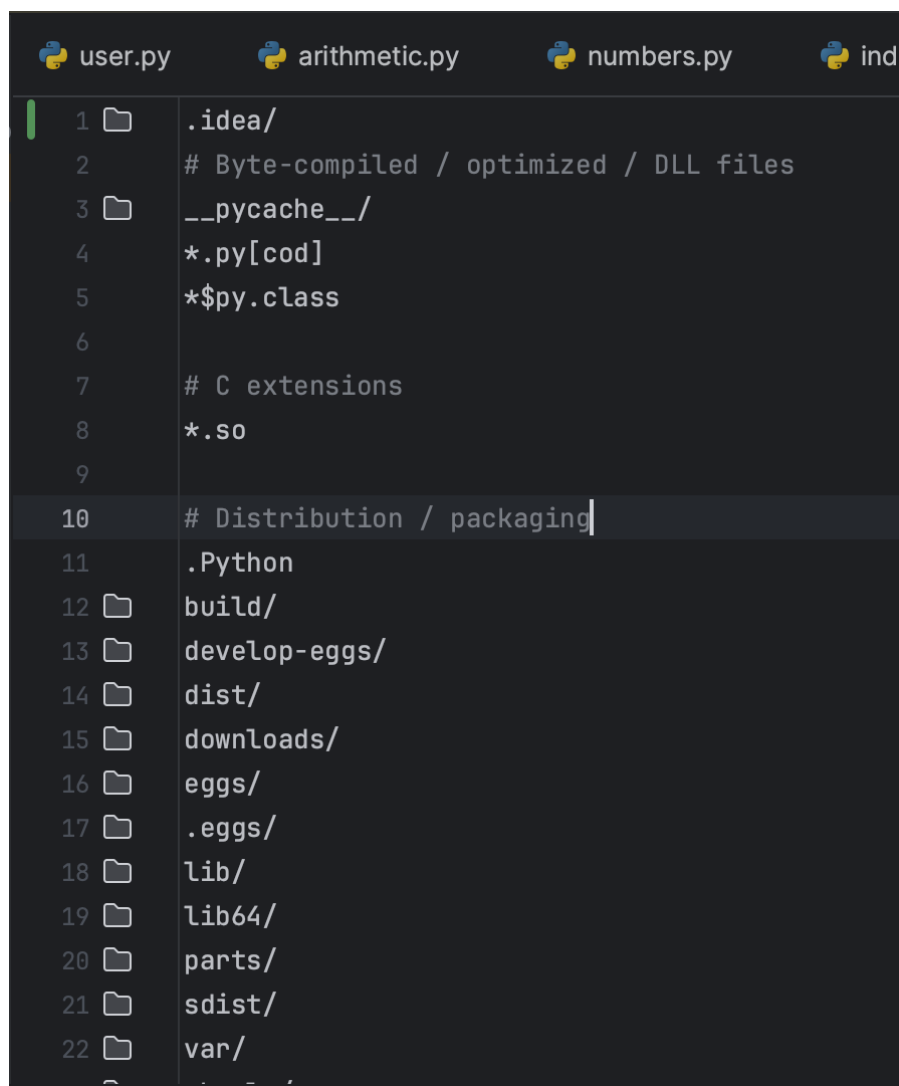
Рисунок 1 – Создание репозитория

2. Клонировал репозиторий на рабочий ПК.

```
Sashka@DESKTOP-U4RPSBI MINGW64 ~/Documents/GitHub
$ git clone https://github.com/SashkaHacker/lab10.git
Cloning into 'lab10'...
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.
```

Рисунок 2 – Клонирование репозитория

3. Дополнил файл .gitignore необходимыми инструкциями.

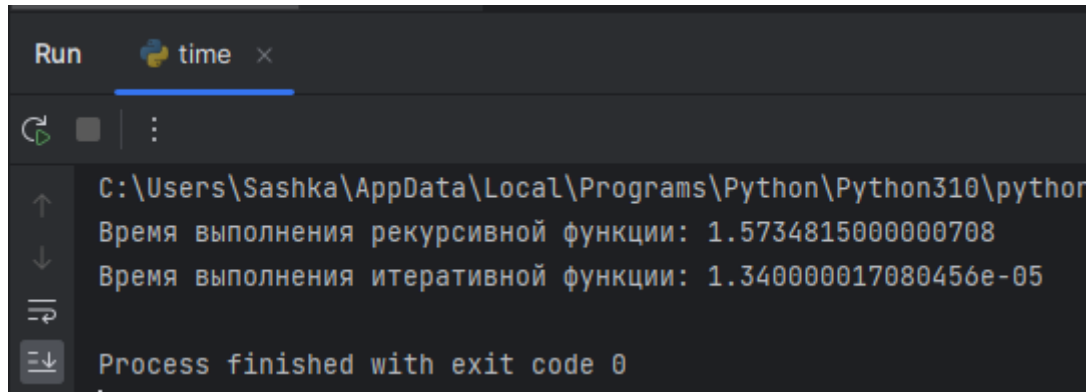


The image shows a code editor window with a dark theme. At the top, there are four tabs: 'user.py', 'arithmetic.py', 'numbers.py', and 'ind'. The active tab is 'user.py'. The editor displays the content of a '.gitignore' file. The text is as follows:

```
1  .idea/
2  # Byte-compiled / optimized / DLL files
3  __pycache__/
4  *.py[cod]
5  *$py.class
6
7  # C extensions
8  *.so
9
10 # Distribution / packaging
11
12 .Python
13 build/
14 develop-eggs/
15 dist/
16 downloads/
17 eggs/
18 .eggs/
19 lib/
20 lib64/
21 parts/
22 sdist/
23 var/
```

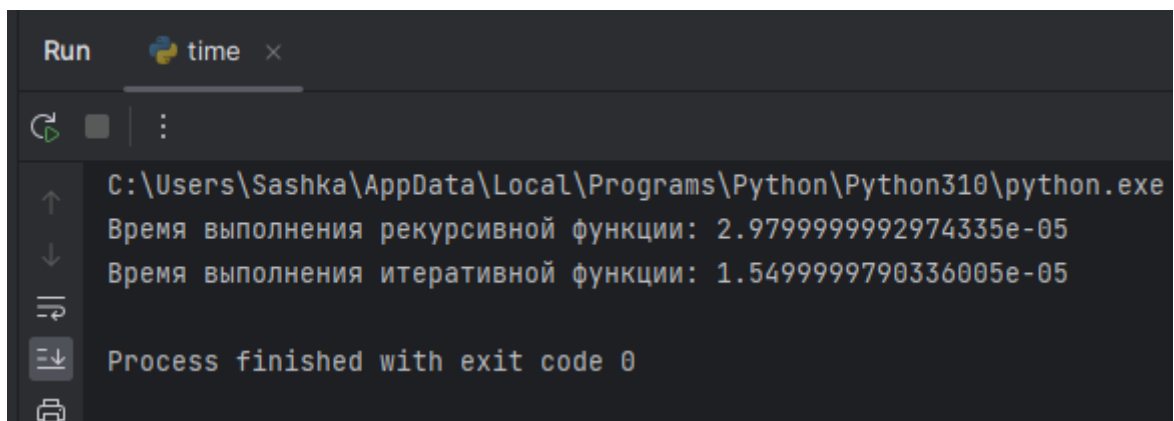
Рисунок 3 – Файл .gitignore

4. Выполнение замеров скорости вычисления итеративной и рекурсивной версий функций factorial и fib при вычислении от числа 30.



```
Run time x
C:\Users\Sashka\AppData\Local\Programs\Python\Python310\python.exe
Время выполнения рекурсивной функции: 1.57348150000000708
Время выполнения итеративной функции: 1.340000017080456e-05
Process finished with exit code 0
```

Рисунок 4 – Время вычисления числа Фибоначчи

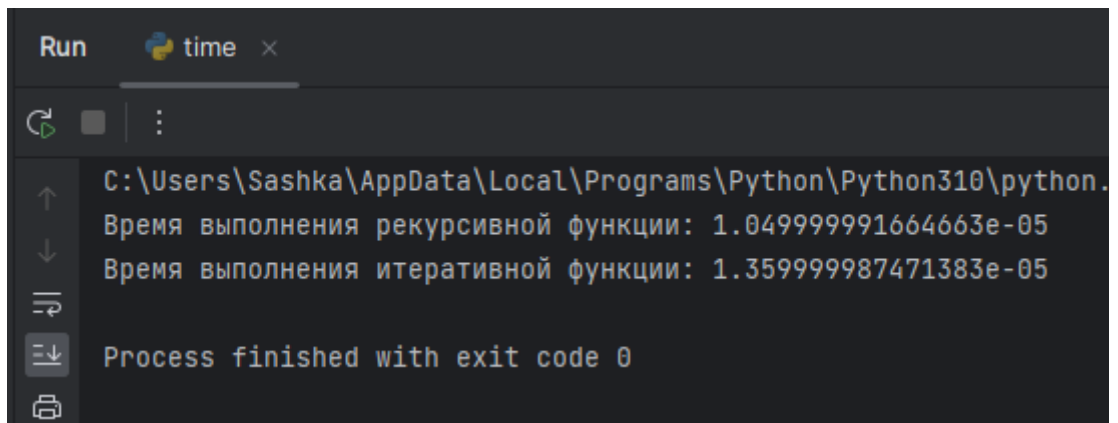


```
Run time x
C:\Users\Sashka\AppData\Local\Programs\Python\Python310\python.exe
Время выполнения рекурсивной функции: 2.979999992974335e-05
Время выполнения итеративной функции: 1.5499999790336005e-05
Process finished with exit code 0
```

Рисунок 5 – Время вычисление факториала

5. Во сколько раз измениться скорость работы рекурсивных версий функций factorial и fib при использовании декоратора lru\_cache?

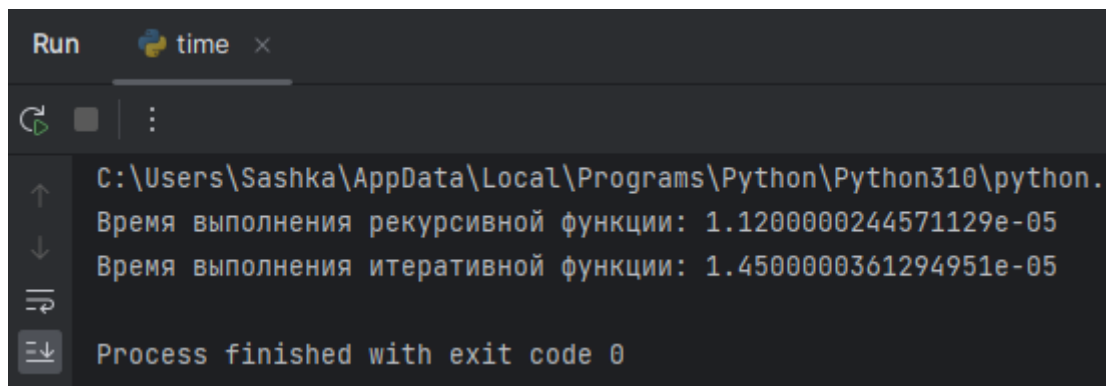
Время вычисления числа Фибоначчи для рекурсивной функции с использованием декоратора lru\_cache уменьшилось в 1.5 раза.



```
Run  time x
C:\Users\Sashka\AppData\Local\Programs\Python\Python310\python.
Время выполнения рекурсивной функции: 1.049999991664663e-05
Время выполнения итеративной функции: 1.359999987471383e-05
Process finished with exit code 0
```

Рисунок 6 – Время вычисления числа Фиббоначи

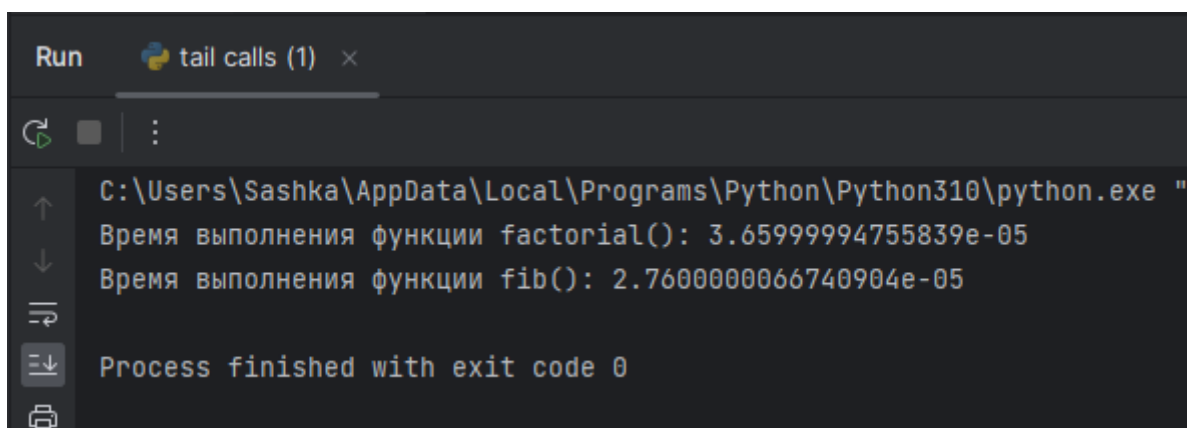
Время вычисления факториала от числа для рекурсивной функции с использованием декоратора `lru_cache` уменьшилось почти в 3 раза.



```
Run  time x
C:\Users\Sashka\AppData\Local\Programs\Python\Python310\python.
Время выполнения рекурсивной функции: 1.1200000244571129e-05
Время выполнения итеративной функции: 1.4500000361294951e-05
Process finished with exit code 0
```

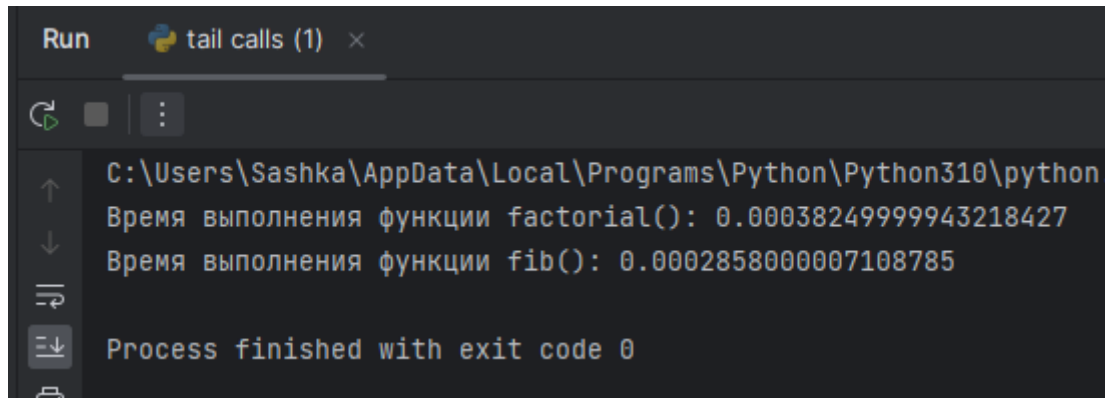
Рисунок 7 – Время вычисления факториала

6. Проработка примера с оптимизацией хвостовых вызовов, время выполнения функций от числа 30:



```
Run  tail calls (1) x
C:\Users\Sashka\AppData\Local\Programs\Python\Python310\python.exe "
Время выполнения функции factorial(): 3.65999994755839e-05
Время выполнения функции fib(): 2.7600000066740904e-05
Process finished with exit code 0
```

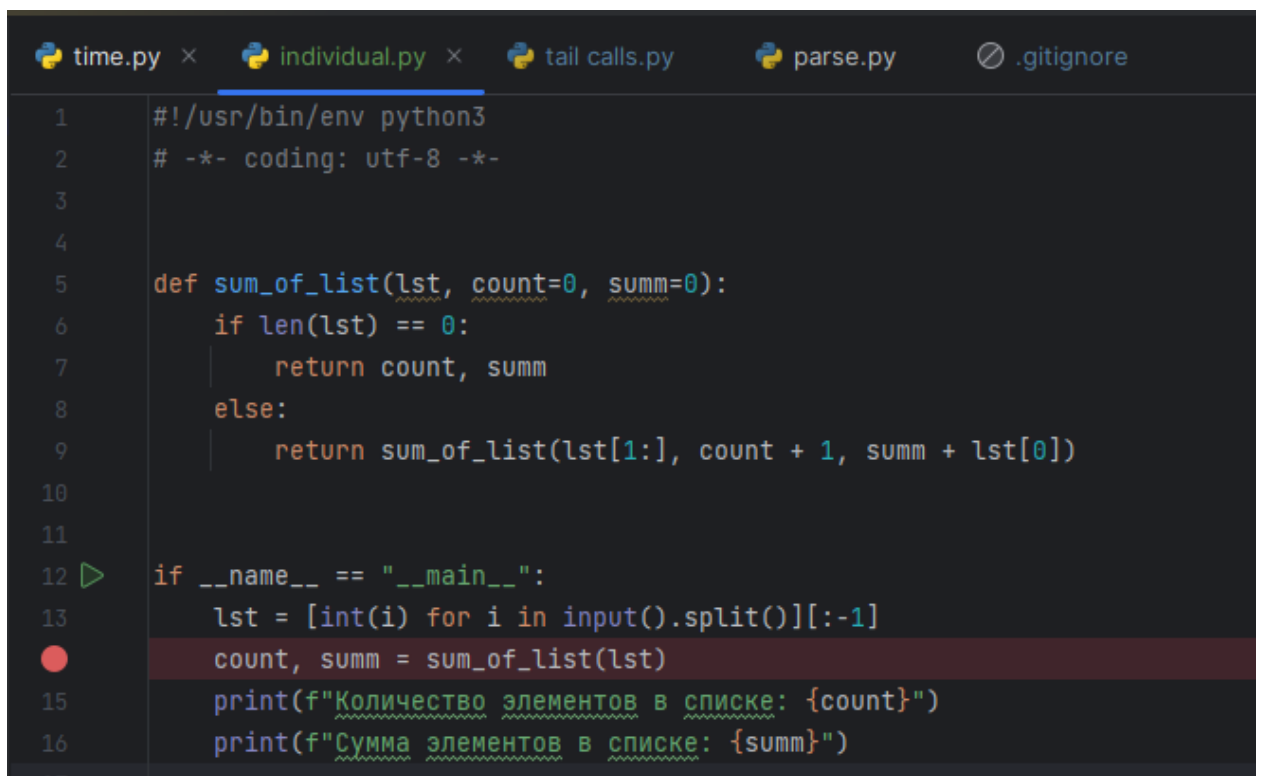
Рисунок 8 – Время выполнения функций без использования интроспекции стека



```
Run tail calls (1) x
C:\Users\Sashka\AppData\Local\Programs\Python\Python310\python
Время выполнения функции factorial(): 0.00038249999943218427
Время выполнения функции fib(): 0.0002858000007108785
Process finished with exit code 0
```

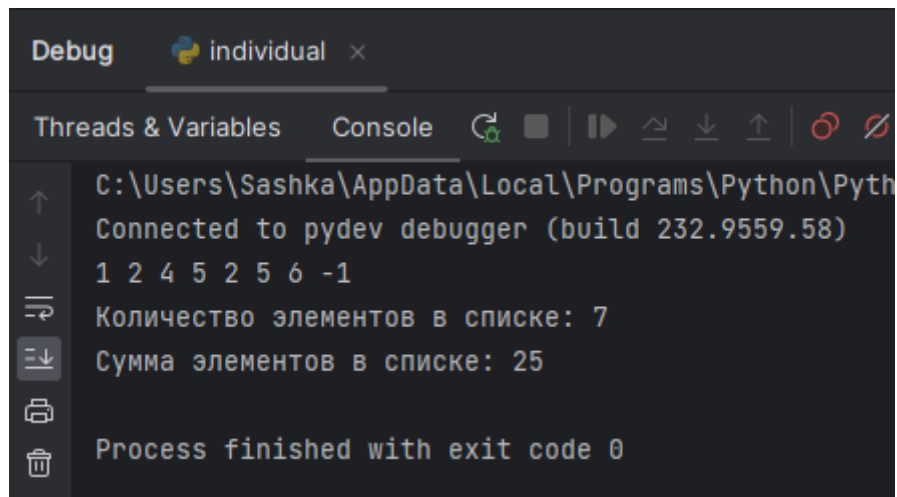
Рисунок 9 – Время выполнения функций с использованием интроспекции стека

## 7. Выполнение индивидуального задания (вариант-11):



```
time.py x individual.py x tail calls.py parse.py .gitignore
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4
5  def sum_of_list(lst, count=0, summ=0):
6      if len(lst) == 0:
7          return count, summ
8      else:
9          return sum_of_list(lst[1:], count + 1, summ + lst[0])
10
11
12  if __name__ == "__main__":
13      lst = [int(i) for i in input().split()][: -1]
14      count, summ = sum_of_list(lst)
15      print(f"Количество элементов в списке: {count}")
16      print(f"Сумма элементов в списке: {summ}")
17
```

Рисунок 10 – Код программы



```
Debug individual x
Threads & Variables Console
C:\Users\Sashka\AppData\Local\Programs\Python\Python38\python.exe
Connected to pydev debugger (build 232.9559.58)
1 2 4 5 2 5 6 -1
Количество элементов в списке: 7
Сумма элементов в списке: 25
Process finished with exit code 0
```

Рисунок 11 – Вывод программы

Ответы на контрольные вопросы:

1. Рекурсия используется в программировании для решения задач, которые можно разбить на более мелкие подзадачи того же типа. Основное свойство рекурсии — это возможность решить задачу, разбив ее на меньшие подзадачи, каждая из которых может быть решена таким же образом. Рекурсия может упростить сложные задачи, разбив их на более мелкие, более управляемые части. Рекурсивный код может быть более читаемым и понятным, чем итеративный код.
2. База рекурсии определяет условие остановки для рекурсивной функции, обеспечивая прекращение рекурсии, когда выполняется определенное условие. Базовый случай в рекурсии определяет условие остановки для рекурсивной функции, гарантируя, что рекурсия прекращается, когда выполняется определенное условие.
3. Стек программы используется для хранения информации о вызовах функций в программе. Каждый раз, когда вызывается функция, все аргументы функции и локальные переменные сохраняются, помещаясь на стек. Когда функция завершает свое выполнение, вся эта память может быть освобождена, просто возвращая стек обратно к состоянию, в котором он был до вызова

функции. Стек программы также используется при рекурсивных вызовах функций. Когда функция вызывает саму себя, создается новый стековый кадр с данными функции, и этот стековый кадр помещается в стек программы.

4. Текущее значение максимальной глубины рекурсии в Python можно получить с помощью функции ``sys.getrecursionlimit()``.

5. Если число рекурсивных вызовов превысит максимальную глубину рекурсии в Python, будет вызвано исключение ``RecursionError: maximum recursion depth exceeded``. Это происходит, чтобы предотвратить бесконечную рекурсию, которая может привести к переполнению стека и сбою Python.

6. Максимальную глубину рекурсии в Python можно изменить с помощью функции ``sys.setrecursionlimit()``. Например, ``sys.setrecursionlimit(1500)`` устанавливает максимальную глубину рекурсии на 1500.

7. Декоратор ``lru_cache`` в Python используется для кэширования результатов функций с использованием стратегии "Least Recently Used" (LRU), или "Наименее недавно использованный". Это позволяет ускорить выполнение функций, сохраняя результаты часто используемых или недавних вызовов функций.

8. Хвостовая рекурсия — это рекурсивная функция, в которой рекурсивный вызов является последним оператором, который выполняется функцией. Оптимизация хвостовых вызовов позволяет функции повторно использовать существующий стековый кадр для рекурсивного вызова, устраняя необходимость в новом стековом кадре. Эта оптимизация возможна только в том случае, если рекурсивный вызов является последней операцией в функции, отсюда и название "хвостовой вызов".