

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

Институт цифрового развития  
Кафедра инфокоммуникаций

**«Рекурсия в языке Python»**

**ОТЧЕТ**  
**по лабораторной работе №12**  
**дисциплины**  
**«Основы программной инженерии»**

Выполнила:

Кувшин Ирина Анатольевна

2 курс, группа ПИЖ-б-о-21-1,

011.03.04 «Программная инженерия»,

направленность (профиль) «Разработка

и сопровождение программного

обеспечения», очная форма обучения

---

(подпись)

Проверил:

---

(подпись)

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

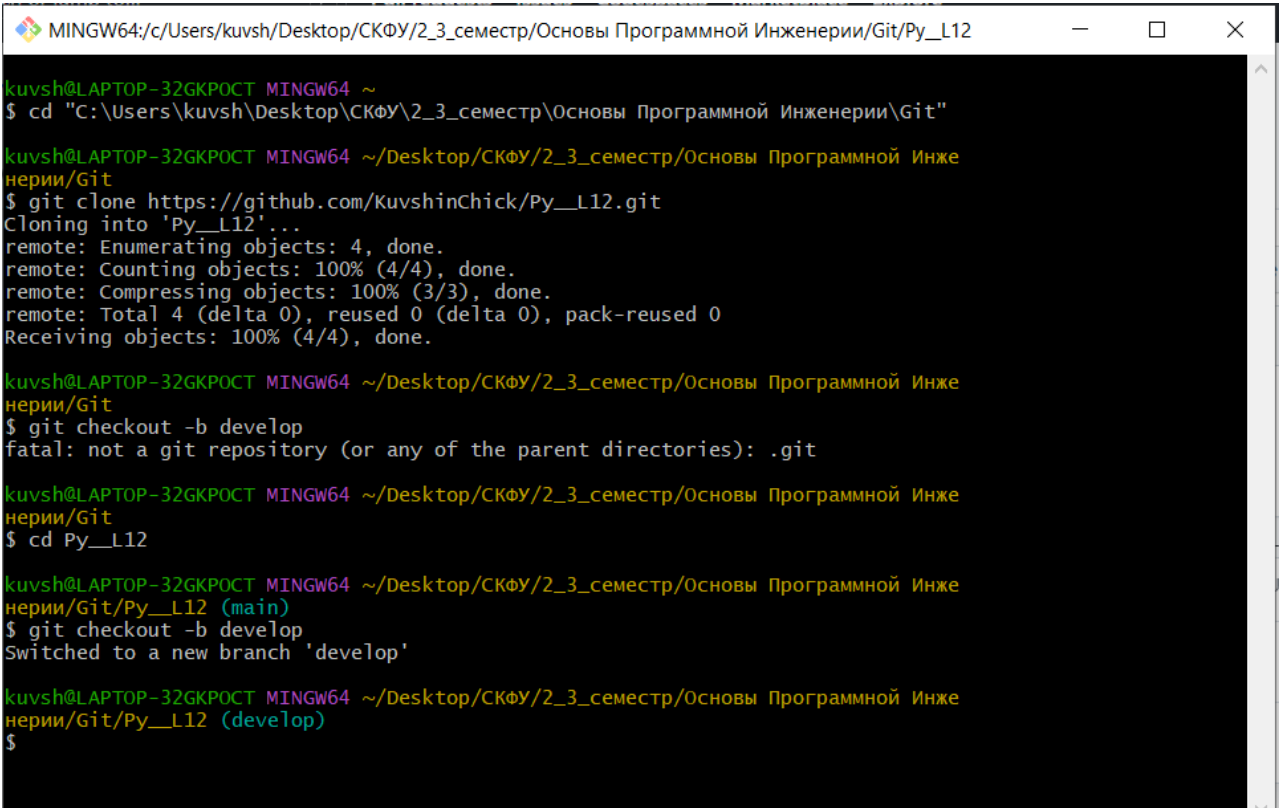
Ставрополь, 2022 г.

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

**Ссылка на репозиторий:** [https://github.com/KuvshinChick/Py\\_\\_L12.git](https://github.com/KuvshinChick/Py__L12.git)

### Ход работы:

1. Изучить теоретический материал работы.
2. Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT и язык программирования Python.
3. Выполните клонирование созданного репозитория.
4. Дополните файл .gitignore необходимыми правилами для работы с IDE PyCharm.
5. Организуйте свой репозиторий в соответствии с моделью ветвления git-flow.



```
MINGW64:/c:/Users/kuvsh/Desktop/СКФУ/2_3_семестр/Основы Программной Инженерии/Git/Py_L12
kuvsh@LAPTOP-32GKPOCT MINGW64 ~
$ cd "C:\Users\kuvsh\Desktop\СКФУ\2_3_семестр\Основы Программной Инженерии\Git"

kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_3_семестр/Основы Программной Инженерии/Git
$ git clone https://github.com/KuvshinChick/Py__L12.git
Cloning into 'Py__L12'...
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (4/4), done.

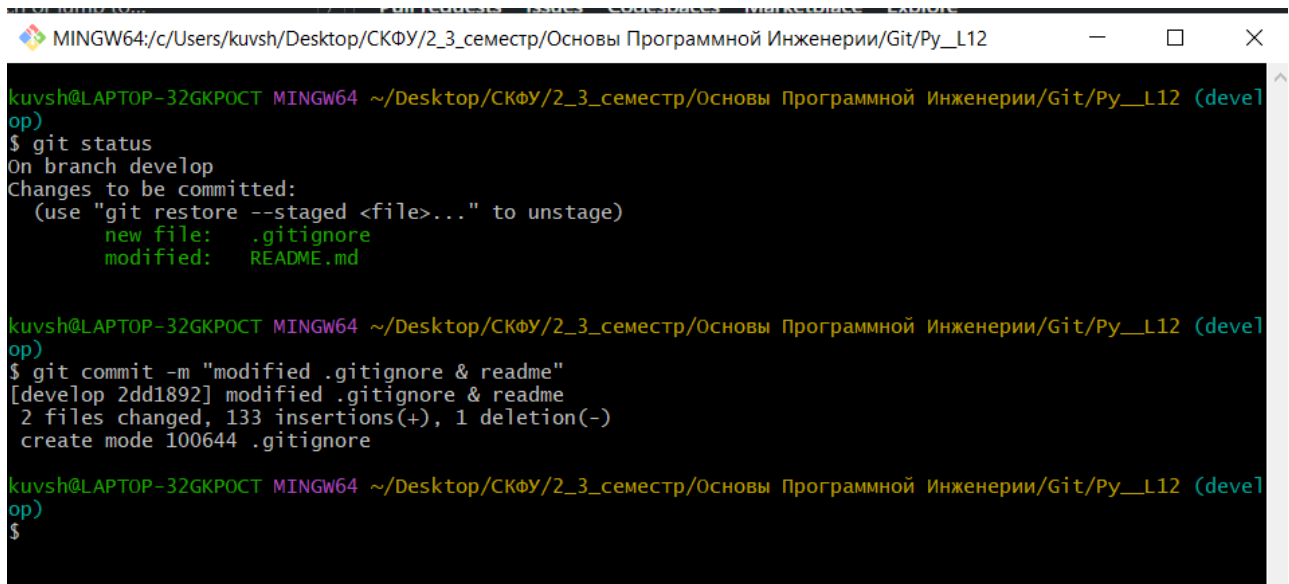
kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_3_семестр/Основы Программной Инженерии/Git
$ git checkout -b develop
fatal: not a git repository (or any of the parent directories): .git

kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_3_семестр/Основы Программной Инженерии/Git
$ cd Py__L12

kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_3_семестр/Основы Программной Инженерии/Git/Py__L12 (main)
$ git checkout -b develop
Switched to a new branch 'develop'

kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_3_семестр/Основы Программной Инженерии/Git/Py__L12 (develop)
$
```

Рисунок 121.1 – Клонирование репозитория и создание ветки develop

A screenshot of a terminal window with a black background and green text. The window title is 'MINGW64: c:/Users/kuvsh/Desktop/СКФУ/2\_3\_семестр/Основы Программной Инженерии/Git/Py\_L12'. The terminal shows the following commands and output:

```
kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_3_семестр/Основы Программной Инженерии/Git/Py_L12 (develop)$ git status
On branch develop
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   .gitignore
        modified:   README.md

kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_3_семестр/Основы Программной Инженерии/Git/Py_L12 (develop)$ git commit -m "modified .gitignore & readme"
[develop 2dd1892] modified .gitignore & readme
 2 files changed, 133 insertions(+), 1 deletion(-)
 create mode 100644 .gitignore

kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_3_семестр/Основы Программной Инженерии/Git/Py_L12 (develop)$
```

Рисунок 12.2 – Обновление .gitignore и readme

6. Создайте проект PyCharm в папке репозитория.

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 измениться скорость работы рекурсивных версий функций `factorial` и `fib` при использовании декоратора `lru_cache`? Приведите в отчет и обоснуйте полученные результаты.

`timeit` — Измеряет время выполнения небольших фрагментов код

`timeit.timeit(stmt='pass', setup='pass', timer=<default timer>,`

`number=1000000, globals=None)`

Создать `Timer` сущность с переданным оператором, `setup` кодом и `timer` функцией с последующим запуском её методом `timeit()` с `number` выполнений. Необязательный аргумент `globals` указывает пространство имён для выполнения кода.

Изменено в версии 3.5: Добавлен необязательный параметр `globals`.

Функция `lru_cache` предназначена для мемоизации, т.е. кэширует результат в памяти. Она используется в качестве декоратора функции, вызовы которой нужно сохранить в памяти вплоть до значения параметра `maxsize` (по умолчанию 128).

Декоратор `lru_cache` подходит для рекурсивных или постоянно вычисляющих функций.

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

import timeit
from functools import lru_cache

def factorial_iter(n):
    product = 1
    while n > 1:
        product *= n
        n -= 1
    return product

def factorial_recurse(n):
    if n == 0:
        return 1
    elif n == 1:
        return 1
    else:
        return n * factorial_recurse(n - 1)

@lru_cache
def factorial_rec_lru(n):
    if n == 0:
        return 1
    elif n == 1:
        return 1
    else:
        return n * factorial_recurse(n - 1)

def fib_iter(n):
    a, b = 0, 1
    while n > 0:
        a, b = b, a + b
        n -= 1
    return a

def fib_recurse(n):
    if n == 0 or n == 1:
        return n
    else:
        return fib_recurse(n - 2) + fib_recurse(n - 1)

@lru_cache
def fib_rec_lru(n):
    if n == 0 or n == 1:
        return n
    else:
        return fib_rec_lru(n - 2) + fib_rec_lru(n - 1)

if __name__ == '__main__':
    print("***** Time for factorial *****")
    print("Time for iterative version:")
    print(f'{timeit.timeit(lambda: factorial_iter(50), number=10000)},\n')
    print("Time for recurse version:")
    print(f'{timeit.timeit(lambda: factorial_recurse(50), number=10000)},\n')
    print("Time for recurse_lru version:")
    print(f'{timeit.timeit(lambda: factorial_rec_lru(50), number=10000)},\n')
    print("***** Time for fib *****")
    print("Time for iterative version:")
    print(f'{timeit.timeit(lambda: fib_iter(15), number=10000)},\n')
```

```

print("Time for recurse version:")
print(f'{timeit.timeit(lambda: fib_recurse(15), number=10000)},\n')
print("Time for recurse_lru version:")
print(timeit.timeit(lambda: fib_rec_lru(15), number=10000))

```

Рисунок 12.3 – Код программы – примера

```

C:\Users\kuvsh\Desktop\CK0V\2_3_семестр\Основы Программной Инженерии\Git\Py_L12\PyCharm\venv\Scripts\python.exe "C:\Users\kuvsh\Desktop\CK0V\2_3_семестр\Основы Программной Инженерии\Git\Py_L12\PyCharm\venv\Scripts\python.exe"
***** Time for factorial *****
Time for iterative version:
0.08387989993207157,

Time for recurse version:
0.10250139993149787,

Time for recurse_lru version:
0.0013016000157222152,

***** Time gor fib *****
Time for iterative version:
0.012690900010056794,

Time for recurse version:
2.2719471999444067,

Time for recurse_lru version:
0.0012293000472709537

Process finished with exit code 0

```

Рисунок 12.4 – Результат работы программы – примера

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.

```

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

import timeit
import sys

class TailRecurseException(Exception):
    def __init__(self, args, kwargs):
        self.args = args
        self.kwargs = kwargs

def tail call optimized(g):
    # Эта программа показывает работу декоратора, который производит
    # оптимизацию хвостового вызова. Он делает это, вызывая исключение, если оно является
    # его прародителем, и перехватывает исключения, чтобы подделать оптимизацию
    # хвоста.
    # Эта функция не работает, если функция декоратора не использует хвостовой
    # вызов.

    def func(*args, **kwargs):
        f = sys._getframe()
        if f.f_back and f.f_back.f_back and f.f_back.f_back.f_code == f.f_code:
            raise TailRecurseException(args, kwargs)
        else:

```

```

        while True:
            try:
                return g(*args, **kwargs)
            except TailRecurseException as e:
                args = e.args
                kwargs = e.kwargs

    func.__doc__ = g.__doc__
    return func

def factorial(n, acc=1):
    if n == 0:
        return acc
    return factorial(n - 1, n * acc)

def fib_recurse(n):
    if n == 0 or n == 1:
        return n
    else:
        return fib_recurse(n - 2) + fib_recurse(n - 1)

@tail_call_optimized
def factorial_opt(n, acc=1):
    if n == 0:
        return acc

    return factorial(n - 1, n * acc)

@tail_call_optimized
def fib_opt(i, current=0, next=1):
    if i == 0:
        return current
    else:
        return fib_opt(i - 1, next, current + next)

if __name__ == '__main__':
    print("***** Time for factorial *****")
    print("Время работы кода без использования интроспекции:")
    print(f'{timeit.timeit(lambda: factorial(500), number=10000)}\n')
    print("Время работы кода с использованием интроспекции:")
    print(f'{timeit.timeit(lambda: factorial_opt(500), number=10000)}\n')
    print("***** Time for fib *****")
    print("Время работы кода без использования интроспекции:")
    print(f'{timeit.timeit(lambda: fib_recurse(15), number=10000)}\n')
    print("Время работы кода с использованием интроспекции:")
    print(f'{timeit.timeit(lambda: fib_opt(15), number=10000)}\n')

```

Рисунок 12.5 – Код программы – примера

```
ex_2 x
"C:\Users\kuvsh\Desktop\СКФУ\2_3_семестр\Основы Программной Инженерии\Git\Py_L12\PyCharm\venv\Scr:
***** Time for factorial *****
Время работы кода без использования интроспекции:
1.5634408000623807

Время работы кода с использованием интроспекции:
1.6602699999930337

***** Time gor fib *****
Время работы кода без использования интроспекции:
2.2882149999495596
Время работы кода с использованием интроспекции:
0.2588794999755919

Process finished with exit code 0
```

Рисунок 12.6 – Результат работы программы – примера

9. Выполните индивидуальные задания. Приведите в отчете скриншоты работы программ решения индивидуального задания.

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

def matched(st, i, counter):
    if counter < 0:
        return "Недопустимая комбинация"
    while i < len(st):
        if st[i] == '(':
            counter += 1
            return matched(st, i + 1, counter)
        elif st[i] == ')':
            counter -= 1
            return matched(st, i + 1, counter)
        else:
            counter += 0
            return matched(st, i + 1, counter)
    if counter == 0:
        return "Ок"
    return "Недопустимая комбинация"

if __name__ == '__main__':
    s = input("Введите строку: ")
    print(matched(s, 0, 0))
```

Рисунок 12.5 – Код программы

```
ind x
"C:\Users\kuvsh\Desktop\СКФУ\2_3_семестр\Основы
Введите строку: ((( )))
Ок
Process finished with exit code 0
```

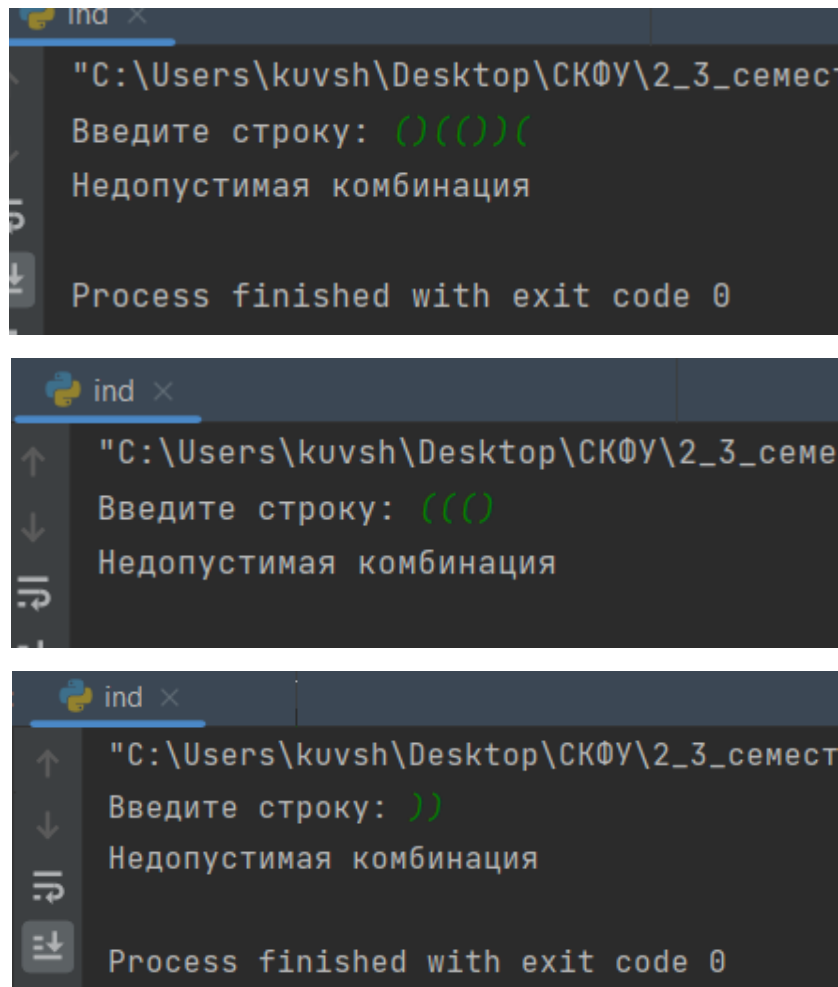


Рисунок 12.6 – Результат работы программы

10. Зафиксируйте сделанные изменения в репозитории.
11. Добавьте отчет по лабораторной работе в формате PDF в папку doc репозитория. Зафиксируйте изменения.
12. Выполните слияние ветки для разработки с веткой master / main.
13. Отправьте сделанные изменения на сервер GitHub.
14. Отправьте адрес репозитория GitHub на электронный адрес преподавателя.



## Контрольные вопросы

1. Для чего нужна рекурсия?

Функция может содержать вызов других функций. В том числе процедура может вызвать саму себя.

2. Что называется базой рекурсии?

У рекурсии, как и у математической индукции, есть база — аргументы, для которых значения функции определены

3. Самостоятельно изучите что является стеком программы. Как используется стек программы при вызове функций?

Максимальная глубина рекурсии ограничена. Точно можно рассчитывать на 10000 вложенных вызовов, некоторые интерпретаторы допускают и больше, но для большинства из них 100000 вызовов — за пределами возможностей. Существуют автоматические оптимизации, помогающие избежать переполнения стека вызовов («оптимизация хвостовой рекурсии»), но они ещё не поддерживаются везде и работают только для простых случаев.

4. Как получить текущее значение максимальной глубины рекурсии в языке Python?

Функция `sys.getrecursionlimit()` возвращает текущее значение предела рекурсии, максимальную глубину стека интерпретатора Python.

5. Что произойдет если число рекурсивных вызовов превысит максимальную глубину рекурсии в языке Python?

Ошибка `RuntimeError`

6. Как изменить максимальную глубину рекурсии в языке Python?

С помощью функции `sys.setrecursionlimit()` модуля `sys`

7. Каково назначение декоратора `lru_cache` ?

Декоратор `@lru_cache()` модуля `functools` оборачивает функцию с переданными в нее аргументами и запоминает возвращаемый результат соответствующий этим аргументам. Такое поведение может сэкономить время и ресурсы, когда дорогая или связанная с вводом/выводом функция периодически вызывается с одинаковыми аргументами.

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. Оптимизация хвостовой рекурсии выглядит так:

```
class recursion(object):
    "Can call other methods inside..."
    def __init__(self, func):
        self.func = func

    def __call__(self, *args, **kwargs):
        result = self.func(*args, **kwargs)
        while callable(result): result = result()
        return result

    def call(self, *args, **kwargs):
        return lambda: self.func(*args, **kwargs)

@recursion
def sum_natural(x, result=0):
    if x == 0:
        return result
    else:
        return sum_natural.call(x - 1, result + x)

# Даже такой вызов не заканчивается исключением
# RuntimeError: maximum recursion depth exceeded
print(sum_natural(1000000))
```