

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

(подпись)

Руководитель практики:
Воронкин Р.А., кандидат
технических наук, доцент кафедры
инфокоммуникаций

(подпись)

Отчет защищен с оценкой _____

Дата защиты _____

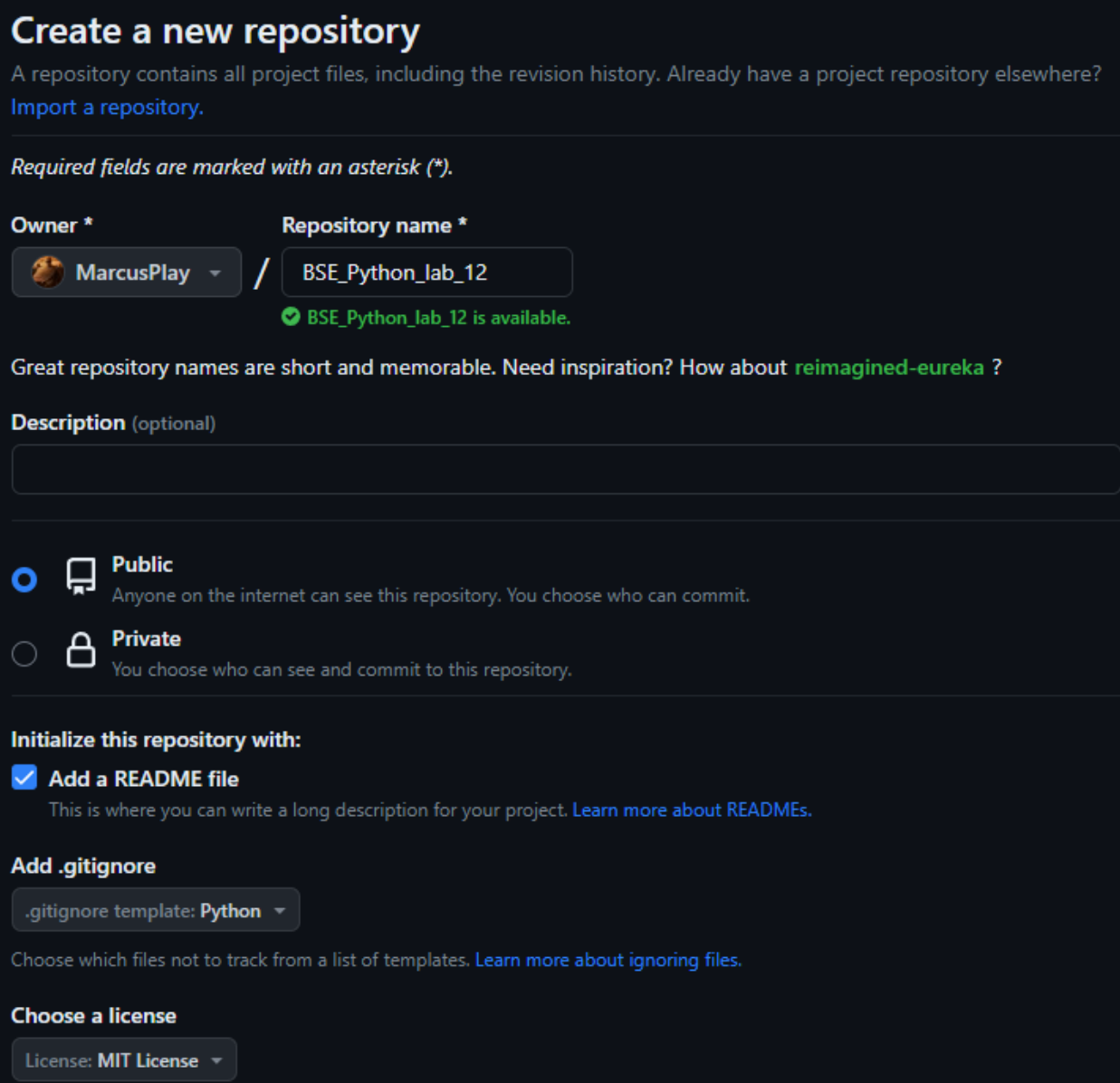
Ставрополь, 2023 г.

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.

Методика и порядок выполнения работы


1. Изучить теоретический материал работы.
2. Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT и язык программирования Python.



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 *  MarcusPlay / Repository name * BSE_Python_lab_12

✔ BSE_Python_lab_12 is available.

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

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**

3. Выполните клонирование созданного репозитория.

```
m4gomedov@Kaskad ~/BasicSoftwareEngineering$ git clone https://github.com/MarcusPlay/BSE_Python_lab_12.git
Cloning into 'BSE_Python_lab_12'...
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (5/5), done.
```

4. Дополните файл .gitignore необходимыми правилами для работы с IDE PyCharm.

5. Организуйте свой репозиторий в соответствие с моделью ветвления git-flow.

```
m4gomedov@Kaskad ~/BasicSoftwareEngineering/BSE_Python_lab_12 > main git checkout -b develop
Switched to a new branch 'develop'
m4gomedov@Kaskad ~/BasicSoftwareEngineering/BSE_Python_lab_12 > develop |
```

6. 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.

```
import timeit
from functools import lru_cache
import sys

sys.set_int_max_str_digits(0)
sys.setrecursionlimit(10000)

# Итеративная версия факториала
def factorial_iterative(n):
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result

# Рекурсивная версия факториала
def factorial_recursive(n):
    if n == 0 or n == 1:
        return 1
    return n * factorial_recursive(n - 1)

# Итеративная версия чисел Фибоначчи
def fib_iterative(n):
    a, b = 0, 1
    for _ in range(n):
        a, b = b, a + b
    return a

# Рекурсивная версия чисел Фибоначчи
def fib_recursive(n):
    if n == 0:
```

```

        return 0
    elif n == 1:
        return 1
    else:
        return fib_recursive(n - 1) + fib_recursive(n - 2)

# Декоратор lru_cache для рекурсивных функций
@lru_cache(maxsize=None)
def factorial_recursive_cached(n):
    if n == 0 or n == 1:
        return 1
    return n * factorial_recursive_cached(n - 1)

@lru_cache(maxsize=None)
def fib_recursive_cached(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib_recursive_cached(n - 1) + fib_recursive_cached(n - 2)

def main(x, y):
    print(f"\nfactorial_iterative({x}) ----->")
    {timeit.timeit(str(factorial_iterative(x)), number=10000)}")
    print(f"factorial_recursive({x}) ----->")
    {timeit.timeit(str(factorial_recursive(x)), number=10000)}")
    print(f"factorial_recursive_cached({x}) ->")
    {timeit.timeit(str(factorial_recursive_cached(x)), number=10000)}")
    print(f"\nfib_iterative({y}) ----->")
    {timeit.timeit(str(fib_iterative(y)), number=10000)}")
    print(f"fib_recursive({y}) ----->")
    {timeit.timeit(str(fib_recursive(y)), number=10000)}")
    print(f"fib_recursive_cached({y}) ----->")
    {timeit.timeit(str(fib_recursive_cached(y)), number=10000)}"\n")

if __name__=="__main__":
    x = int(input())
    y = int(input())
    main(x, y)

```

```

x m4gomedov@Kaskad ~/BasicSoftwareEngineering/BSE_Python_lab_12 develop python3 task_1.py
4700
40

factorial_iterative(4700) -----> 0.000161278000632592
factorial_recursive(4700) -----> 0.00016886200046428712
factorial_recursive_cached(4700) -> 0.00013500800014298875

fib_iterative(40) -----> 0.00013392599976214115
fib_recursive(40) -----> 0.0001378030001433217
fib_recursive_cached(40) -----> 0.0001754849999997532

```

7. 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.

```

import sys
from functools import lru_cache
import timeit

sys.setrecursionlimit(10000)

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

@lru_cache()
def factorial_tail_recursive(n, acc=1):
    if n == 0:
        return acc
    else:
        return factorial_tail_recursive(n-1, n*acc)

if __name__ == "__main__":
    x = int(input())

    time_recursive = timeit.timeit(str(factorial(x)), number=10000)
    time_tail_recursive = timeit.timeit(str(factorial_tail_recursive(x)), number=10000)

    print(f"Время выполнения обычной рекурсивной функции от {x}: {time_recursive}")
    print(f"Время выполнения оптимизированной хвостовой рекурсивной функции от {x}: {time_tail_recursive}")

```

```

m4gomedov@Kaskad ~/BasicSoftwareEngineering/BSE_Python_lab_12 develop python3 task_2.py
500
Время выполнения обычной рекурсивной функции от 500: 0.00013434599986794638
Время выполнения оптимизированной хвостовой рекурсивной функции от 500: 0.00013423600012174575

```

8. Выполните индивидуальные задания. Приведите в отчете скриншоты работы программ решения индивидуального задания. Дан список `X` из вещественных чисел. Найти минимальный элемент списка,

используя вспомогательную рекурсивную функцию, находящую минимум среди последних элементов списка X , начиная с n -го.

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

def find_min_recursive(X, n):
    if n == len(X) - 1:
        return X[n]

    rest_min = find_min_recursive(X, n + 1)

    if X[n] < rest_min:
        return X[n]
    else:
        return rest_min

if __name__ == "__main__":
    X = list(map(float, input("Введите значения списка через пробел:\n").split))
    n = int(input("Введите индекс с которого будет вестись поиск: "))
    result = find_min_recursive(X, n)
    print(f"Минимальный элемент списка, начиная с {n}-го: {result}")
```

```
X m4gomedov@Kaskad ~/BasicSoftwareEngineering/BSE_Python_lab_12 develop
Введите значения списка через пробел:
1 2 3 4 5 6 7 1 2 34
Введите индекс с которого будет вестись поиск: 4
Минимальный элемент списка, начиная с 4-го: 1.0
```

9. Зафиксируйте сделанные изменения в репозитории.

```
X • m4gomedov@Kaskad ~/BasicSoftwareEngineering/BSE_Python_lab_12 develop + git commit -m "added individual_task"
[develop 163dd04] added individual_task
1 file changed, 21 insertions(+)
create mode 100644 individual_task.py
```

10. Добавьте отчет по лабораторной работе в *формате PDF* в папку *doc* репозитория.

11. Зафиксируйте изменения.

12. Выполните слияние ветки для разработки с веткой *master* / *main*.

```

Imran@Kaskad MINGW64 ~/Desktop/Work/BSE_Python_lab_12 (main)
$ git merge develop
Updating fb2d6fe..163dd04
Fast-forward
 individual_task.py | 21 +++++
 task_1.py           | 73 +++++
 task_2.py           | 38 +++++
 3 files changed, 132 insertions(+)
 create mode 100644 individual_task.py
 create mode 100644 task_1.py
 create mode 100644 task_2.py

```

13. Отправьте сделанные изменения на сервер GitHub.

```

Imran@Kaskad MINGW64 ~/Desktop/Work/BSE_Python_lab_12 (main)
$ git push -u origin main
Enumerating objects: 8, done.
Counting objects: 100% (8/8), done.
Delta compression using up to 12 threads
Compressing objects: 100% (7/7), done.
Writing objects: 100% (7/7), 2.41 KiB | 821.00 KiB/s, done.
Total 7 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), done.
To https://github.com/MarcusPlay/BSE_Python_lab_12.git
    fb2d6fe..163dd04  main -> main
branch 'main' set up to track 'origin/main'.

```

14. Отправьте адрес репозитория GitHub на электронный адрес преподавателя.

Вопросы для защиты работы

1. Для чего нужна рекурсия?

Рекурсия — это подход в программировании, при котором функция вызывает саму себя. Она применяется для решения задач, которые могут быть разбиты на более простые подзадачи.

2. Что называется базой рекурсии?

База рекурсии — это условие, при котором функция перестает вызывать саму себя и возвращает конкретное значение. Это условие необходимо, чтобы избежать бесконечного цикла вызовов.

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

Стек программы представляет собой структуру данных, используемую для хранения информации о вызовах функций в программе. При вызове функции информация о текущем состоянии помещается в вершину стека, а при завершении функции эта информация удаляется.

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

Для получения текущего значения максимальной глубины рекурсии в Python можно использовать модуль `sys` и атрибут `getrecursionlimit()`.

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

Если число рекурсивных вызовов превысит максимальную глубину рекурсии в Python, произойдет ошибка `"RecursionError: maximum recursion depth exceeded"`.

6. Как изменить максимальную глубину рекурсии в языке Python?

Максимальную глубину рекурсии в Python можно изменить с помощью функции `sys.setrecursionlimit()`. Однако, изменение этого значения требует осторожности, так как слишком большое значение может привести к проблемам с памятью или крашам программы.

7. Каково назначение декоратора `lru_cache`?

Декоратор `lru_cache` используется для кеширования результатов вызова функции с определенными аргументами. Он помогает избежать повторных вычислений, сохраняя результаты в памяти.

8. Что такое хвостовая рекурсия? Как проводится оптимизация хвостовых вызовов?

Хвостовая рекурсия — это случай, когда рекурсивный вызов является последней операцией в функции. В некоторых языках программирования, таких как Scheme, хвостовая рекурсия может быть оптимизирована компилятором для уменьшения использования стека. В Python оптимизация

хвостовых вызовов не поддерживается стандартным интерпретатором CPython, но она может быть полезной в других языках.