

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 №2_9
дисциплины
«Основы программной инженерии»

Выполнил:

Разворотников Денис Сергеевич
2 курс, группа ПИЖ-б-о-21-1,
09.03.04 «Программная инженерия»,
направленность (профиль) «Разработка
и сопровождение программного
обеспечения», очная форма обучения

(подпись)

Проверил:

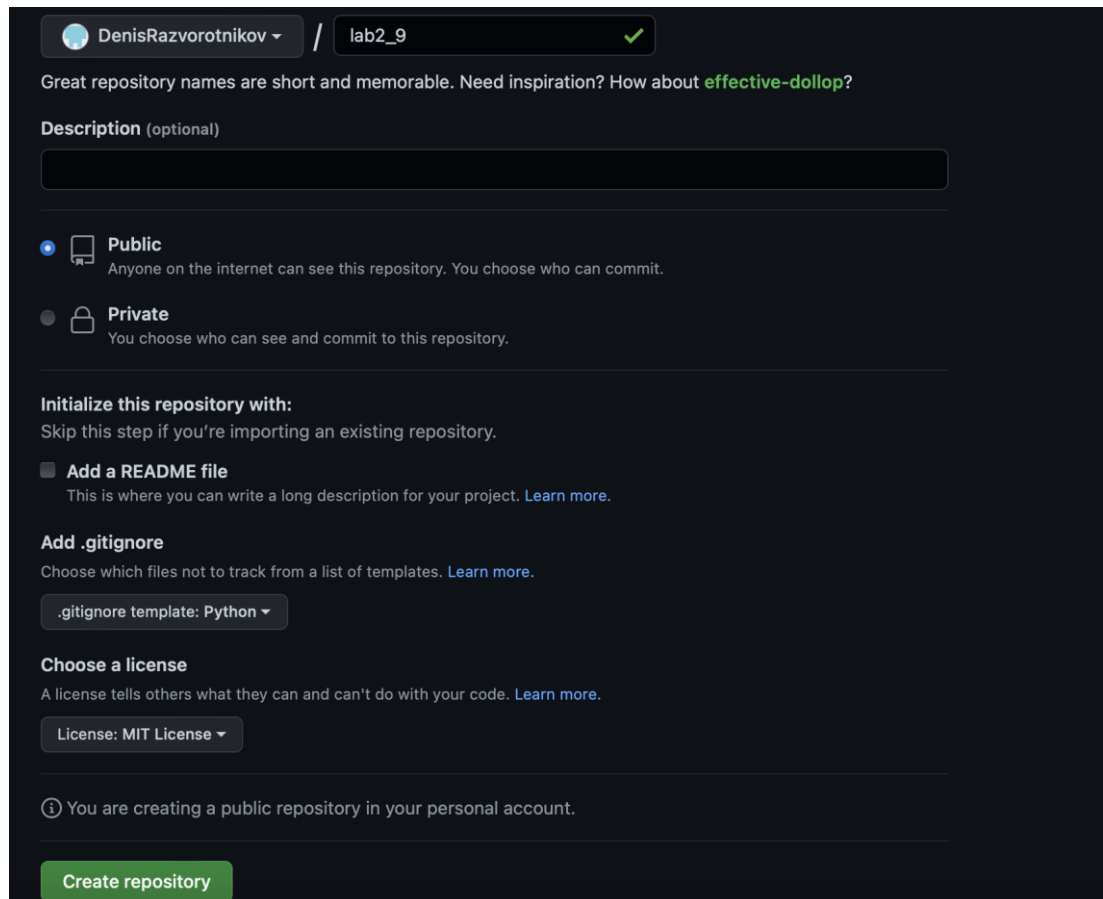
(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь 2022

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.



DenisRazvorotnikov / lab2_9 ✓

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

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

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

Create repository

Рисунок 1 – Создание репозитория

```
denisrazvorotnikov@Air-Denis desktop % git clone https://github.com:denisrazvorotnikov/lab2_9.git
Cloning into 'lab2_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.
denisrazvorotnikov@Air-Denis desktop %
```

Рисунок 2 – Клонирование репозитория

```
Process finished with exit code 0
```

Рисунок 3 – Результат работы 1 примера

```
Process finished with exit code 0
```

Рисунок 4 – Результат работы 2

```
/usr/local/bin/python3 /Users/denisrazvorotnikov/Desktop/lab2_9/4.  
  
Process finished with exit code 0
```

Рисунок 5 – 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.

```
/usr/local/bin/python3 /Users/denisrazvorotnikov/Desktop/lab2_9/4.  
Результат рекурсивного факториала: 1.0874937288463116e-05  
Результат рекурсивного числа Фибоначи: 1.0124989785254002e-05  
Результат итеративного факториала: 1.0416959412395954e-05  
Результат итеративного числа Фибоначи: 1.0207993909716606e-05  
Результат факториала с декоратором: 1.0584015399217606e-05  
Результат числа Фибоначи с декоратором: 1.0332907550036907e-05  
  
Process finished with exit code 0
```

Рисунок 6 – 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.

```
/usr/local/bin/python3 /Users/denisrazvorotnikov/Desktop/lab2_9/5.py
Результат факториала: 1.5583005733788013e-05
Результат числа Фибоначи: 1.4959019608795643e-05
Результат факториала с интроспекцией стека: 1.4999997802078724e-05
Результат числа Фибоначи с интроспекцией стека: 1.4916062355041504e-05

Process finished with exit code 0
```

Рисунок 7 – 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

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))
```

```
/usr/local/bin/python3 /Users/denisrazvorotnikov/Desktop/
3 4 5
[[], [5], [4], [4, 5], [3], [3, 5], [3, 4], [3, 4, 5]]

Process finished with exit code 0
```

Рисунок 8 – Результат работы программы

```
[denisrazvorotnikov@Air-Denis lab2_9 % git add .  
[denisrazvorotnikov@Air-Denis lab2_9 % git commit -m "Add file"  
[main 8d1ac32] Add file  
12 files changed, 223 insertions(+)  
create mode 100644 .idea/.gitignore  
create mode 100644 .idea/inspectionProfiles/profiles_settings.xml  
create mode 100644 .idea/lab2_9.iml  
create mode 100644 .idea/misc.xml  
create mode 100644 .idea/modules.xml  
create mode 100644 .idea/vcs.xml  
create mode 100644 1.py  
create mode 100644 2.py  
create mode 100644 3.py  
create mode 100644 4.py  
create mode 100644 5.py  
create mode 100644 idz.py  
denisrazvorotnikov@Air-Denis lab2_9 %
```

Рисунок 9 – Коммит и пуш изменений

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 `RunTime`.

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