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

Институт перспективной инженерии
Департамент цифровых, робототехнических систем и электроники

ОТЧЕТ
ПО ПРАКТИЧЕСКОЙ РАБОТЕ №6
дисциплины «Программирование на Python»

Вариант 20

Выполнил:
Скоробогатов Виктор Андреевич
2 курс, группа ИВТ-б-о-24-1,
09.03.01 «Информатика и вычислительная техника»,
направленность (профиль) «Программное обеспечение средств вычислительной техники и автоматизированных систем», очная форма обучения

_____
(подпись)

Руководитель практики:
Воронкин Роман Александрович,
доцент департамента цифровых, робототехнических систем и электроники института перспективной инженерии

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2025 г.

Тема: Работа с функциями в языке Python

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

Ход работы

Ссылка на репозиторий:

https://github.com/LostsSs78/NCFU-Python_26-02-11-21-00

Для выполнения работы был создан новый общедоступный репозиторий на GitHub, использующий лицензию MIT и язык программирования Python, далее он был склонирован на персональный компьютер, а также дополнен файл .gitignore:

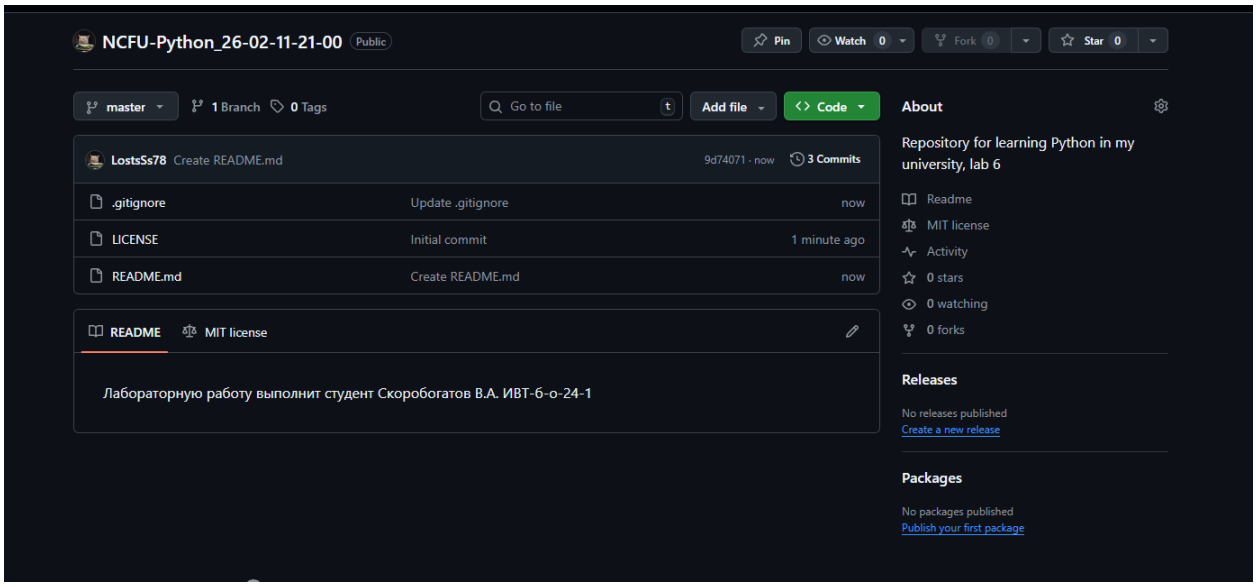


Рисунок 1. Репозиторий на GitHub

Далее были и проработаны примеры, созданы отдельные модули для каждого примера и зафиксированы изменения в репозитории:

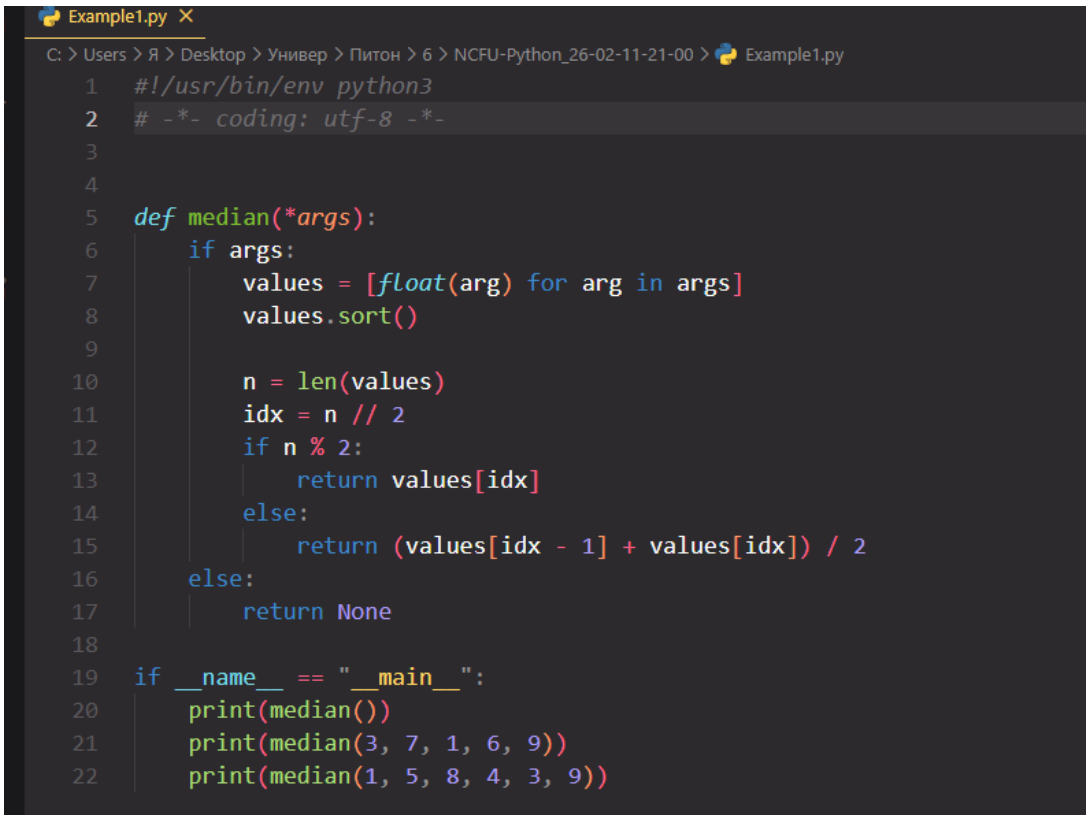


Рисунок 2. Пример 1

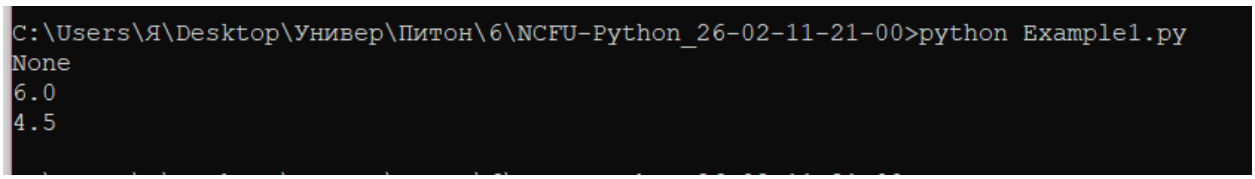
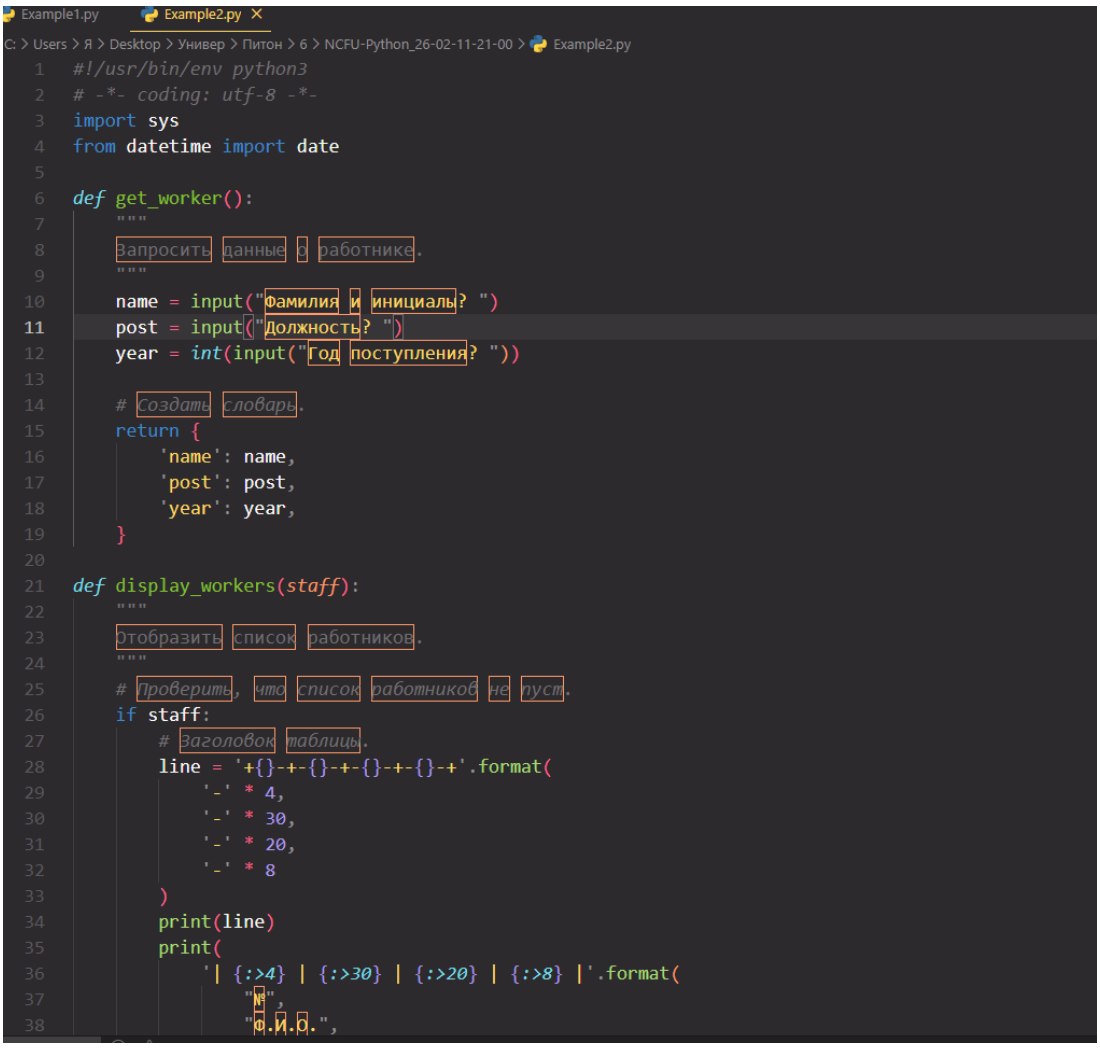


Рисунок 3. Результат выполнения кода примера 1 с разными входными данными



```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import sys
from datetime import date

def get_worker():
    """
    Запросить данные о работнике.
    """
    name = input("Фамилия и инициалы? ")
    post = input("Должность? ")
    year = int(input("Год поступления? "))

    # Создать словарь.
    return {
        'name': name,
        'post': post,
        'year': year,
    }

def display_workers(staff):
    """
    Отобразить список работников.
    """
    # Проверить, что список работников не пуст.
    if staff:
        # Заголовок таблицы.
        line = '+{}-+-{}-+-{}-+-{}-+'.format(
            '-' * 4,
            '-' * 30,
            '-' * 20,
            '-' * 8
        )
        print(line)
        print(
            '| {:>4} | {:>30} | {:>20} | {:>8} |'.format(
                "№",
                "Ф.И.О.",
```

Рисунок 4. Пример 2

```
C:\Users\Я\Desktop\Универ\Питон\6\NCFU-Python_26-02-11-21-00>python Example2.py
>> help
Список команд:

add - добавить работника;
list - вывести список работников;
select <стаж> - запросить работников со стажем;
help - отобразить справку;
exit - завершить работу с программой.
>> add
Фамилия и инициалы? Иванов Иван Иванович
Должность? зав. отделения
Год поступления? 2021
>> list
+-----+--------------------------------+----------------------+----------+
|    № |                         Ф.И.О. |            Должность |      Год |
+-----+--------------------------------+----------------------+----------+
|    1 | Иванов Иван Иванович           | зав. отделения       |     2021 |
+-----+--------------------------------+----------------------+----------+
>> select 1
+-----+--------------------------------+----------------------+----------+
|    № |                         Ф.И.О. |            Должность |      Год |
+-----+--------------------------------+----------------------+----------+
|    1 | Иванов Иван Иванович           | зав. отделения       |     2021 |
+-----+--------------------------------+----------------------+----------+
>>
```

Рисунок 5. Результат выполнения кода примера 2 с разными входными данными

```
raindance@DESKTOP-33LH9V8 MINGW64 /d/python/4/NCFU-Python_03-12-24-13-14 (master
)
$ git add .

raindance@DESKTOP-33LH9V8 MINGW64 /d/python/4/NCFU-Python_03-12-24-13-14 (master
)
$ git commit -m "Examples done"
[master c7efb41] Examples done
 3 files changed, 75 insertions(+)
 create mode 100644 Example1.py
 create mode 100644 Example2.py
 create mode 100644 Example3.py
```

Рисунок 6. Фиксация изменений в репозитории

Индивидуальное задание 1: составить программу с использованием списков и словарей для решения задачи. Оформить каждое действие в виде отдельной функции. Передавать данные через параметры функции и не использовать глобальные переменные. Использовать словарь, содержащий следующие ключи: фамилия и инициалы; номер группы; успеваемость (список из пяти элементов). Написать программу, выполняющую следующие действия: ввод с клавиатуры данных в список, состоящий из словарей заданной структуры; записи должны быть упорядочены по возрастанию номера группы; вывод на дисплей фамилий и номеров групп для всех студентов, включенных в массив, если средний балл студента больше 4.0; если таких студентов нет, вывести соответствующее сообщение.

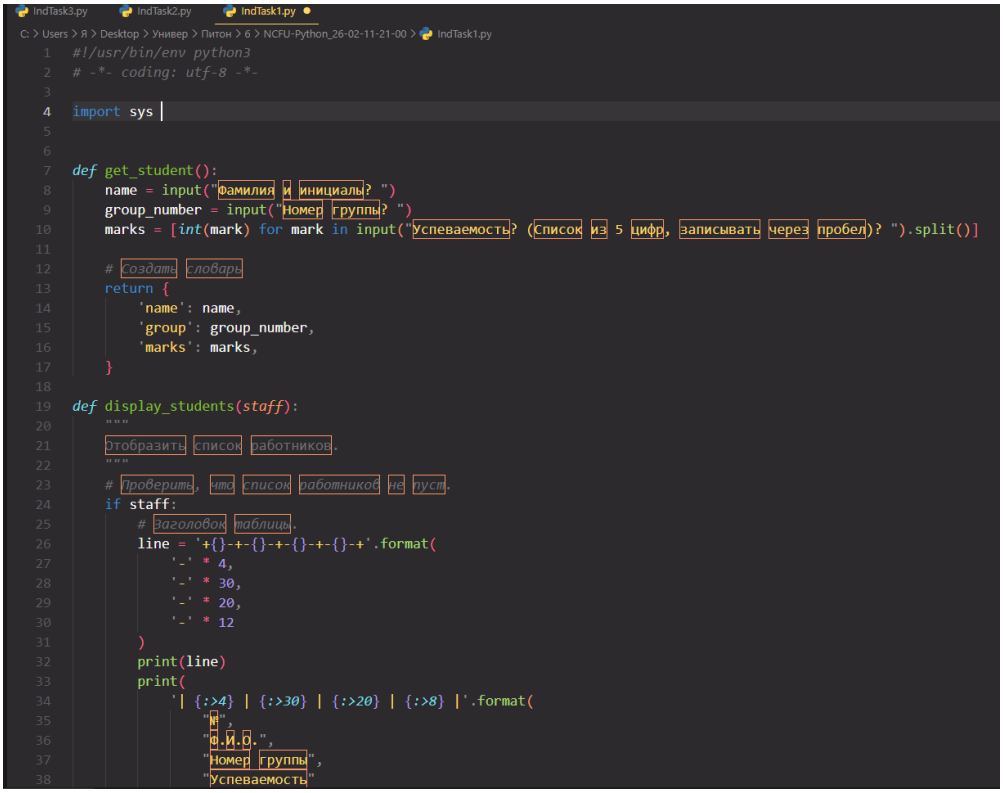


Рисунок 7. Код для индивидуального задания 1

```
C:\Users\Я\Desktop\Универ\Питон\6\NCFU-Python_26-02-11-21-00>python IndTask1.py
>> add
Фамилия и инициалы? Скоробогатов В.А.
Номер группы? 1
Успеваемость? (Список из 5 цифр, записывать через пробел)? 4 5 4 3 3
>> add
Фамилия и инициалы? Ромащенко Д.П.
Номер группы? 3
Успеваемость? (Список из 5 цифр, записывать через пробел)? 5 5 5 5 5
>> add
Фамилия и инициалы? Оразов Т.И.
Номер группы? 1
Успеваемость? (Список из 5 цифр, записывать через пробел)? 4 4 5 3 4
>> list
+-----+------------------------------+----------------------+--------------+
|   № |                      Ф.И.О. |         Номер группы | Успеваемость |
+-----+------------------------------+----------------------+--------------+
|   1 | Скоробогатов В.А.            | 1                    | 4 5 4 3 3 |
|   2 | Оразов Т.И.                  | 1                    | 4 4 5 3 4 |
|   3 | Ромащенко Д.П.               | 3                    | 5 5 5 5 5 |
+-----+------------------------------+----------------------+--------------+
>> select
+-----+------------------------------+----------------------+--------------+
|   № |                      Ф.И.О. |         Номер группы | Успеваемость |
+-----+------------------------------+----------------------+--------------+
|   1 | Ромащенко Д.П.               | 3                    | 5 5 5 5 5 |
+-----+------------------------------+----------------------+--------------+
```

Рисунок 8. Результат выполнения кода для индивидуального задания 1

Индивидуальное задание 2: Создайте функцию debug_call(*args, **kwargs), которая возвращает строку отчёта о вызове функции:

- количество позиционных аргументов,
- количество именованных аргументов,
- типы всех переданных значений.

C: > Users > Я > Desktop > Универ > Питон > 6 > NCFU-Python_26-02-11-21-00 > IndTask2.py

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

def debug_call(*args, **kwargs):
    args_count = len(args)
    kwargs_count = len(kwargs)

    types_list = []

    # Добавляем типы позиционных аргументов
    for arg in args:
        types_list.append(type(arg).__name__)

    # Добавляем типы именованных аргументов
    for value in kwargs.values():
        types_list.append(type(value).__name__)

    types_str = ", ".join(types_list)

    return f"Args: {args_count}, Kwargs: {kwargs_count}, Types: {types_str}"


# Пример использования
if __name__ == "__main__":
    # Тестовые вызовы
    result1 = debug_call(1, "строка", 3.14, name="Иван", age=25)
    print(result1)

    result2 = debug_call(10, 20, 30, a=1, b=[2, 3], c={"x": 5})
    print(result2)
```

Рисунок 9. Код для индивидуального задания 2

```
C:\Users\Я\Desktop\Универ\Питон\6\NCFU-Python_26-02-11-21-00>python IndTask2.py
Args: 3, Kwargs: 2, Types: int, str, float, str, int
Args: 3, Kwargs: 3, Types: int, int, int, int, list, dict
```

Рисунок 10. Результат выполнения кода для индивидуального задания 2

Индивидуальное задание 3: Реализуйте функцию remove_keys(data, condition), которая рекурсивно проходит по словарю и удаляет все ключи, для которых condition(key, value) возвращает True.

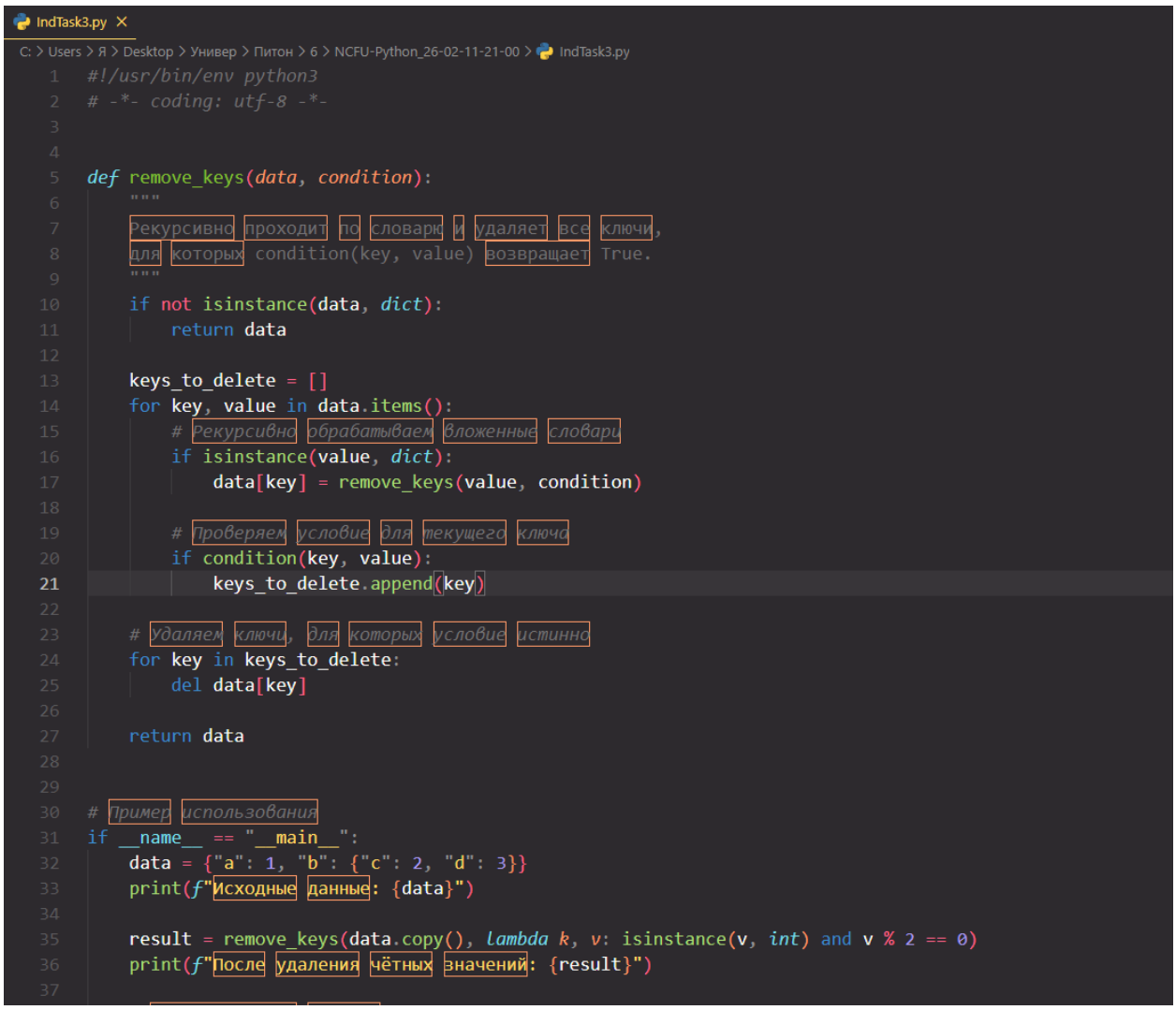


Рисунок 11. Код для индивидуального задания 3

```
C:\Users\Я\Desktop\Универ\Питон\6\NCFU-Python_26-02-11-21-00>python IndTask3.py
Исходные данные: {'a': 1, 'b': {'c': 2, 'd': 3}}
После удаления чётных значений: {'a': 1, 'b': {'d': 3}}

Другой пример: {'x': 10, 'y': 15, 'z': {'p': 20, 'q': 25, 'r': {'m': 30, 'n': 35}}}
После удаления чётных значений: {'y': 15, 'z': {'q': 25, 'r': {'n': 35}}}

Удаление ключей на 'a': {'banana': 3, 'cherry': {'blueberry': 2}}
```

Рисунок 12. Результат выполнения кода для индивидуального задания 3

Контрольные вопросы

1) Функция в программировании представляет собой обособленный участок кода, который можно вызывать, обратившись к нему по имени, которым он был назван. При вызове происходит выполнение команд тела функции.

2) В языке программирования Python функции определяются с помощью оператора def. Ключевое слово return в Python используется для возврата значения из функции.

3) Локальные переменные видны только в локальной области видимости, которой может выступать отдельно взятая функция. Глобальные переменные видны во всей программе. "Видны" значит, известны, доступны. К ним можно обратиться по имени и получить связанное с ними значение.

4) В Питоне позволительно возвращать из функции несколько объектов, перечислив их через запятую после команды return.

5) В программировании функции могут не только возвращать данные, но также принимать их, что реализуется с помощью так называемых параметров, которые указываются в скобках в заголовке функции. Количество параметров может быть любым. Параметры представляют собой локальные переменные, которым присваиваются значения в момент вызова функции.

6) Значение по умолчанию присваивается аргументу с помощью оператора присваивания "=".

7) Python поддерживает интересный синтаксис, позволяющий определять небольшие однострочные функции на лету. Позаимствованные из Lisp, так называемые lambda-функции могут быть использованы везде, где требуется функция. Lambda - это выражение, а не инструкция. По этой причине ключевое слово lambda может появляться там, где синтаксис языка Python не позволяет использовать инструкцию def, - внутри литералов или в вызовах функций.

8) Документирование кода согласно PEP257. Строки документации - строковые литералы, которые являются первым оператором в модуле, функции, классе или определении метода. Такая строка документации становится специальным атрибутом __doc__ этого объекта.

Все модули должны, как правило, иметь строки документации, и все функции, и классы, экспортируемые модулем также должны иметь строки

документации. Публичные методы (в том числе __init__) также должны иметь строки документации. Пакет модулей может быть документирован в __init__ py.

Для согласованности, всегда используйте "*"triple double quotes""" для строк документации. Используйте r"""w triple double quotes"*" , если вы будете использовать обратную косую черту в строке документации.

9) Существует две формы строк документации: однострочная и многострочная.

Одиночные строки документации предназначены для действительно очевидных случаев. Они должны умещаться на одной строке. Однострочная строка документации не должна быть "подписью" параметров функции / метода (которые могут быть получены с помощью интроспекции).

Многострочные строки документации состоят из однострочной строки документации с последующей пустой строкой, а затем более подробным описанием. Первая строка может быть использована автоматическими средствами индексации, поэтому важно, чтобы она находилась на одной строке и была отделена от остальной документации пустой строкой. Первая строка может быть на той же строке, где и открывающие кавычки, или на следующей строке. Вся документация должна иметь такой же отступ, как кавычки на первой строке.

10) Позиционные аргументы в Python — это аргументы, которые передаются функции в строго определённом порядке. Их значения сопоставляются с параметрами по позиции: первому параметру соответствует первый аргумент, второму — второй и так далее.

11) Именованными (ключевыми) аргументами в Python называются аргументы, которым присвоены имена.

12) Оператор "*" позволяет «распаковывать» объекты, внутри которых хранятся некие элементы.

13) *args - это сокращение от «arguments» (аргументы), а **kwargs - сокращение от «keyword arguments» (именованные аргументы). Каждая из

этих конструкций используется для распаковки аргументов соответствующего типа, позволяя вызывать функции со списком аргументов переменной длины.

14) Рекурсия. Функция может содержать вызов других функций. В том числе процедура может вызвать саму себя. Никакого парадокса здесь нет - компьютер лишь последовательно выполняет встретившиеся ему в программе команды и, если встречается вызов процедуры, просто начинает выполнять эту функцию. Без разницы, какая функция дала команду это делать.

15) Утверждение, if n == 0: return 1, называется базовым случаем. Потому что это не рекурсия. Базовый случай необходим, без него вы столкнетесь с бесконечной рекурсией. С учетом сказанного, если у вас есть хотя бы один базовый случай, у вас может быть столько случаев, сколько вы хотите.

16) Стек вызовов (call stack) — это специальная структура данных в памяти, которая работает по принципу LIFO (Last In, First Out — "последним пришел, первым ушел").

При вызове функции:

– В стек помещается кадр стека (stack frame) для этого вызова. Он содержит локальные переменные функции, ее аргументы и адрес возврата (место в программе, куда нужно вернуться после завершения функции).

– Управление передается вызванной функции.

– Если эта функция вызывает другую, процесс повторяется (новый кадр кладется поверх предыдущего).

– Когда функция завершается (достигает return или конца), ее кадр удаляется из стека, и управление возвращается по адресу возврата в предыдущий кадр (вызывающей функции).

17) Чтобы проверить текущие параметры лимита, нужно запустить: sys.getrecursionlimit().

18) Существует предел глубины возможной рекурсии, который зависит от реализации Python. Когда предел достигнут, возникает исключение RuntimeError: RuntimeError: Maximum Recursion Depth Exceeded.

19) Можно изменить предел глубины рекурсии с помощью вызова: sys.setrecursionlimit(limit).

20) Декоратор Lru_cache можно использовать для уменьшения количества лишних вычислений.

21) Хвостовой вызов - это просто вызов рекурсивной функции, который является последней операцией и должна быть выполнена перед возвратом значения. Чтобы было понятно, return foo(n - 1) - это хвост вызова, но return foo(n - 1) + 1 не является (поскольку операция сложения будет последней операцией).

Оптимизация хвостового вызова (TCO) - это способ автоматического сокращения рекурсии в рекурсивных функциях.

В Python нет TCO по нескольким причинам, поэтому для обхода этого ограничения можно использовать другие методы. Выбор используемого метода зависит от варианта использования.

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