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

Институт перспективной инженерии
Департамент цифровых, роботехнических систем и электроники

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №6
дисциплины «Программирование на Python»
Вариант 7

Выполнила:
Еремина Татьяна Евгеньевна
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)

Проверил:
Воронкин Р.А., 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 при работе с функциями при написании программ с помощью языка программирования Python версии 3.x.

Практическая часть:

Исходный репозиторий: https://github.com/TL-hash/labs6.

Пример 1. Разработать функцию для определения медианы значений аргументов функции. Если функции передается пустой список аргументов, то она должна возвращать значение None.

Решение:

1. Проверяем, были ли вообще введены аргументы с помощью условия.
2. Создаём список значений и сортируем его.
3. Определяем середину списка, используя функцию len() и целочисленное деление.
4. Если индекс нечетный, то возвращаем значение с этим индексом, иначе выводим полусумму соседних значений посередине.

Листинг примера 1:

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

def median(*args):
    if args:
        values = [float(arg) for arg in args]
        values.sort()

        n = len(values)
        idx = n // 2
        if n % 2:
            return values[idx]
        else:
            return (values[idx - 1] + values[idx]) / 2

    else:
        return None
```

```
if __name__ == "__main__":
    print(median())
    print(median(3, 7, 1, 6, 9))
    print(median(1, 5, 8, 4, 3, 9))
```

Результат выполнения:

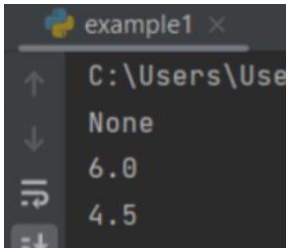

Рисунок 1 - Результат выполнения примера 1

Пример 2. Использовать словарь, содержащий следующие ключи: фамилия и инициалы работника; название занимаемой должности; год поступления на работу. Написать программу, выполняющую следующие действия:

- ввод с клавиатуры данных в список, состоящий из заданных словарей;
- записи должны быть размещены по алфавиту;
- вывод на дисплей фамилий работников, чей стаж работы в организации превышает значение, введенное с клавиатуры;
- если таких работников нет, вывести на дисплей соответствующее сообщение.

Оформить каждое действие в виде отдельной функции.

Решение:

1. Функция get_worker() запрашивает данные о работнике и создаёт словарь.

2. Функция display_workers(staff) отображает список работников: проверяет, не пуст ли список, формирует заголовок таблицы и записи о работниках.

3. Функция select_workers(staff, period) позволяет выбрать работников с заданным стажем, получает на вход текущую дату, формирует список и возвращает результат.

4. Функция main() – главная функция программы.

Листинг примера 2:

```
#!/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.",
                "Должность",
                "Год"
            )
        )
        print(line)

        # Вывести данные о всех сотрудниках.
        for idx, worker in enumerate(staff, 1):
```

```
            print(
                '| {:>4} | {:<30} | {:<20} | {:>8} |'.format(
                    idx,
                    worker.get('name', ''),
                    worker.get('post', ''),
                    worker.get('year', 0)
                )
            )

        print(line)

    else:
        print("Список работников пуст.")


def select_workers(staff, period):
    """
    Выбрать работников с заданным стажем.
    """
    # Получить текущую дату.
    today = date.today()

    # Сформировать список работников.
    result = []
    for employee in staff:
        if today.year - employee.get('year', today.year) >= period:
            result.append(employee)

    # Возвратить список выбранных работников.
    return result


def main():
    """
    Главная функция программы.
    """
    # Список работников.
    workers = []

    # Организовать бесконечный цикл запроса команд.
    while True:
        # Запросить команду из терминала.
        command = input(">>> ").lower()

        # Выполнить действие в соответствие с командой.
        if command == 'exit':
            break

        elif command == 'add':
            # Запросить данные о работнике.
            worker = get_worker()
```

```
            # Добавить словарь в список.
            workers.append(worker)
            # Отсортировать список в случае необходимости.
            if len(workers) > 1:
                workers.sort(key=lambda item: item.get('name', ''))

        elif command == 'list':
            # Отобразить всех работников.
            display_workers(workers)

        elif command.startswith('select '):
            # Разбить команду на части для выделения стажа.
            parts = command.split(' ', maxsplit=1)
            # Получить требуемый стаж.
            period = int(parts[1])

            # Выбрать работников с заданным стажем.
            selected = select_workers(workers, period)
            # Отобразить выбранных работников.
            display_workers(selected)

        elif command == 'help':
            # Вывести справку о работе с программой.
            print("Список команд:\n")
            print("add - добавить работника;")
            print("list - вывести список работников;")
            print("select <стаж> - запросить работников со стажем;")
            print("help - отобразить справку;")
            print("exit - завершить работу с программой.")

        else:
            print(f"Неизвестная команда {command}", file=sys.stderr)


if __name__ == '__main__':
    sys.exit(main())
```

Результат выполнения:

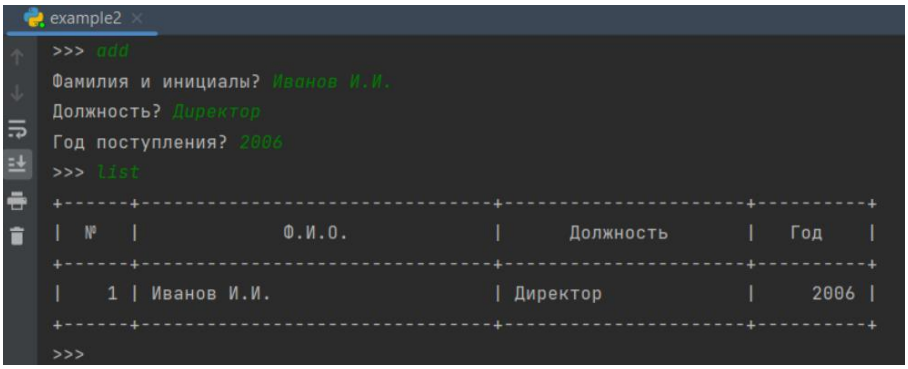

Рисунок 2 - Результат выполнения примера 2

Задание 1 (вариант 7). 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. Номер варианта определяется по согласованию с преподавателем.

Использовать словарь, содержащий следующие ключи: название пункта назначения; номер поезда; время отправления. Написать программу, выполняющую следующие действия: ввод с клавиатуры данных в список, состоящий из словарей заданной структуры; записи должны быть упорядочены по времени отправления поезда; вывод на экран информации о поездах, направляющихся в пункт, название которого введено с клавиатуры; если таких поездов нет, выдать на дисплей соответствующее сообщение.

Решение:

1. Функция get_train() запрашивает данные о поездах, проверяет с помощью try правильность ввода времени и создаёт словарь.
2. Функция display_trains(trains_list) проверяет, пуст список поездов или нет, формирует заголовок таблицы и выводить записи.
3. Функция select_trains(trains_list, target_destination) позволяет выбрать поезда, которые направляются в указанный пункт.
4. Функция main() – главная функция программы, создаёт бесконечный цикл для ввода команд, сопоставляет введённые команды с существующими функциями, выводит список всех команд.

Листинг задания 1:

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

import sys
from datetime import datetime


def get_train():
    """
    Запросить данные о поезде.
    """
    destination = input("Название пункта назначения? ")
```

```
    number = input("Номер поезда? ")
    time_str = input("Время отправления (ЧЧ:ММ)? ")

    try:
        departure_time = datetime.strptime(time_str, "%H:%M").time()
    except ValueError:
        print("Неверный формат времени. Используйте ЧЧ:ММ.", file=sys.stderr)
        return None

    return {
        'destination': destination,
        'number': number,
        'departure_time': departure_time
    }


def display_trains(trains_list):
    """
    Вывести список поездов в виде таблицы.
    """
    if not trains_list:
        print("Список поездов пуст.")
        return

    line = '+-{}-+-{}-+-{}-+'.format(
        '-' * 20,
        '-' * 15,
        '-' * 10
    )
    print(line)
    print(
        '| {:^20} | {:^15} | {:^10} |'.format(
            "Пункт назначения",
            "Номер поезда",
            "Время"
        )
    )
    print(line)

    for train in trains_list:
        print(
            '| {:<20} | {:<15} | {:>10} |'.format(
                train['destination'],
                train['number'],
                train['departure_time'].strftime("%H:%M")
            )
        )

    print(line)


def select_trains(trains_list, target_destination):
```

```
    """
    Выбрать поезда, направляющиеся в указанный пункт назначения.
    """
    return [
        train for train in trains_list
        if train['destination'].lower() == target_destination.lower()
    ]


def main():
    """
    Главная функция программы.
    """
    trains = []

    while True:
        command = input(">>> ").strip().lower()

        if command == 'exit':
            break

        elif command == 'add':
            train = get_train()
            if train is not None:
                trains.append(train)
                trains.sort(key=lambda item: item['departure_time'])

        elif command == 'list':
            display_trains(trains)

        elif command.startswith('select '):
            parts = command.split(' ', maxsplit=1)
            if len(parts) < 2:
                print("Не указано название пункта назначения.", file=sys.stderr)
                continue

            target_destination = parts[1].strip()
            found_trains = select_trains(trains, target_destination)

            if found_trains:
                print(f"\nПоезда, направляющиеся в {target_destination}:")
                for train in found_trains:
                    print(
                        f'                Поезд №{train['number']}, отправление в
{train['departure_time'].strftime('%H:%M')}'
                    )
            else:
                print(f"Поездов в пункт '{target_destination}' не найдено.")

        elif command == 'help':
            print("Список команд:\n")
            print("add - добавить информацию о поезде;")
```

```
            print("list - вывести список всех поездов;")
            print("select <пункт> - показать поезда, идущие в указанный пункт;")
            print("help - отобразить справку;")
            print("exit - завершить работу с программой.")

        else:
            print(f"Неизвестная команда: {command}", file=sys.stderr)


if __name__ == "__main__":
    sys.exit(main())
```

Результат выполнения:

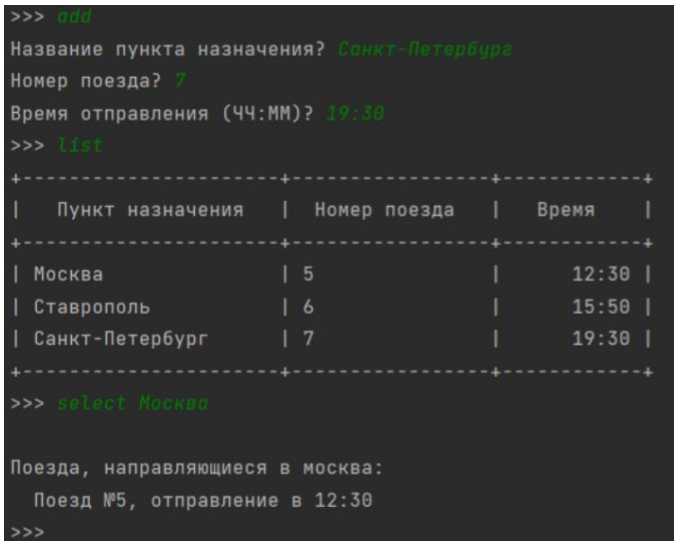

Рисунок 3 - Результат выполнения задания 1

Задание 2 (вариант 7). Создайте функцию user_profile(name, *interests, **details), которая возвращает отформатированную строку с именем, списком интересов и детальной информацией. Если интересов нет – вывести «Интересы не указаны».

Решение:

1. Создаём функцию user_profile(name, *interests, **details), в ней name – обязательный позиционный аргумент (имя пользователя), *interests – собирает все дополнительные позиционные аргументы в кортеж, **details – собирает все именованные аргументы в словарь.

2. Создаём список parts, в который будет добавляться текст частей профиля.

3. Проверяем, были ли введены интересы, если да, то объединяем их через запятую и добавляем сроку interests.

4. Проверяем, details пуст или нет, с помощью генератора строк преобразуем словарь в строку вида "city: Paris, age: 25".

5. Возвращаем строку, в которой объединяем все части через запятую.

Листинг задания 2:

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

def user_profile(name, *interests, **details):
    parts = [f"Имя: {name}"]

    if interests:
        parts.append(f"Интересы: {', '.join(interests)}")
    else:
        parts.append("Интересы не указаны")

    if details:
        details_str = ', '.join(f"{k}: {v}" for k, v in details.items())
        parts.append(f"Дополнительная информация: {details_str}")

    return ", ".join(parts)


if __name__ == "__main__":
    result = user_profile("Alice", "music", "travel", city="Paris", age=25)
    print(result)
```

Результат выполнения:

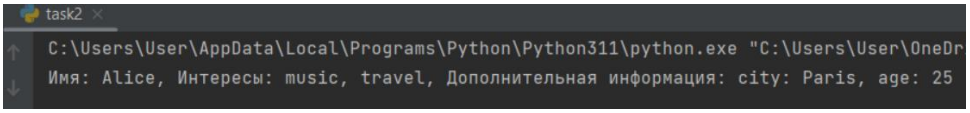

Рисунок 4 - Результат выполнения задания 2

Задание 3 (вариант 7). Напишите рекурсивную функцию count_substring(text, sub), которая считает, сколько раз sub встречается в text, без использования str.count().

Решение:

1. Создаём функцию count_substring(text, sub), записываем начальное условие, если длина подстроки больше длины строки, возвращаем None.

2. Если строка начинается с подстроки, то считаем первое вхождение, затем вызываем функцию рекурсивно, начиная с остатка после подстроки text[len(sub):], чтобы найти неперекрывающиеся последовательности.

3. Если первое вхождение в начале не найдено, рекурсивно вызываем функцию, передавая строку без первого символа: text[1:].

Листинг задания 3:

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

def count_substring(text, sub):
    if len(text) < len(sub):
        return None

    if text.startswith(sub):
        return 1 + count_substring(text[len(sub):], sub)
    else:
        return count_substring(text[1:], sub)


if __name__ == "__main__":
    print(count_substring("banana", "ana"))
    print(count_substring("aaaa", "aa"))
    print(count_substring("ананас", "ана"))
```

Результат выполнения:

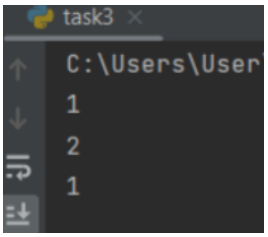

Рисунок 5 - Результат выполнения задания 3

Контрольные вопросы:

1. Каково назначение функций в языке программирования Python?

Функции позволяют инкапсулировать логику, избегать дублирования кода, повышать читаемость и модульность программ. Они принимают входные данные, выполняет вычисления и возвращают результат.

2. Каково назначение операторов def и return?

def – объявляет функцию и задаёт её имя, параметры и тело.

return – завершает выполнение функции и возвращает значение вызывающему коду. Если return отсутствует, функция возвращает None.

3. Каково назначение локальных и глобальных переменных при написании функций в Python?

Локальные переменные существуют только внутри функции и недоступны снаружи. Глобальные переменные объявлены вне функций и доступны везде (но для изменения внутри функции требуется global). Это обеспечивает изоляцию и предотвращает побочные эффекты.

4. Как вернуть несколько значений из функции Python?

В Python можно вернуть кортеж (или список/словарь): return a, b, c.

5. Какие существуют способы передачи значений в функцию?

Позиционные аргументы – передаются по порядку.

Именованные аргументы – передаются как param=value.

6. Как задать значение аргументов функции по умолчанию?

Указать значение при объявлении: def func(a, b=10): ...

7. Каково назначение lambda-выражений в языке Python?

lambda создаёт анонимную (безымянную) функцию в одну строку. Чаще используется как аргумент для функций (map, filter, sort) или в случаях, где функция нужна временно.

8. Как осуществляется документирование кода согласно PEP257?

Согласно PEP 257, каждая функция (и модуль/класс) должна иметь строку документации (docstring) – строку в тройных кавычках сразу после заголовка. Она описывает назначение, параметры, возвращаемое значение.

9. В чем особенность однострочных и многострочных форм строк документации?

Однострочная пишется в одну строку, многострочная содержит краткое описание с подробностями.

10. Какие аргументы называются позиционными в Python?

Аргументы, передаваемые в порядке, соответствующем параметрам функции, без указания имени называются позиционными.

11. Какие аргументы называются именованными в Python?

Именованные аргументы – аргументы, передаваемые в виде имя=значение.

12. Для чего используется оператор *?

Оператор * используется для: распаковки последовательностей: func(*args) и объявления переменного числа позиционных аргументов: def f(*args).

13. Каково назначение конструкций *args и **kwargs?

*args – собирает все дополнительные позиционные аргументы в кортеж. **kwargs – собирает все дополнительные именованные аргументы в словарь.

14. Для чего нужна рекурсия?

Рекурсия применяется для решения задач, которые можно разбить на подзадачи того же типа (например, обход деревьев, вычисление факториала, поиск в структурах).

15. Что называется базой рекурсии?

База рекурсии – это условие, при котором функция не вызывает саму себя, а возвращает результат напрямую. Она предотвращает бесконечную рекурсию.

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

Стек в программировании – это абстрактный тип данных, который работает по принципу LIFO (Last In, First Out): последний вошедший элемент выходит первым. Стек используется при вызове функций для хранения

информации о выполняемых функциях программы. Когда программа вызывает функцию, в стек помещаются:

- адрес возврата (куда вернуться после завершения функции);
- параметры функции;
- локальные переменные функции;
- регистры процессора, которые нужно сохранить.

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

Получить текущее значение максимальной глубины рекурсии в языке Python можно с помощью модуля sys: import sys

print(sys.getrecursionlimit()).

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

Возникнет исключение RecursionError: maximum recursion depth exceeded, и программа аварийно завершится.

19. Как изменить максимальную глубину рекурсии в языке Python?

С помощью sys.setrecursionlimit(n): import sys sys.setrecursionlimit(2000).

20. Каково назначение декоратора lru_cache?

Декоратор functools.lru_cache кэширует результаты вызовов функции по аргументам. Это позволяет автоматически запоминать результаты предыдущих вызовов функции и использовать их повторно.

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

Хвостовая рекурсия – это рекурсия, в которой вызов функции является последней операцией в теле (нет вычислений после возврата).

Для оптимизации можно преобразовать функцию в итеративный код или использовать декоратор lru_cache, который можно использовать для уменьшения количества лишних вычислений.

Вывод: приобретены навыки при работе с функциями при написании программ с помощью языка программирования Python версии 3.x. Были освоены основные принципы работы с функциями: объявление, передача аргументов (включая *args и **kwargs), возврат значений и рекурсия, отработаны навыки документирования в соответствии с PEP 257.