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 ЛАБОРАТОРНОЙ РАБОТЕ №6
дисциплины «Программирование на Python»

Выполнила:
Федорова Дарья Юрьевна
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 Р. А.

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2025 г.

Вариант 25

Ссылка на репозиторий: https://github.com/Vishhh123/Repozitoriy_6

Тема: работа с функциями в языке Python

Цель: приобретение навыков по работе с функциями при написании программ с помощью языка программирования Python.

Ход работы:

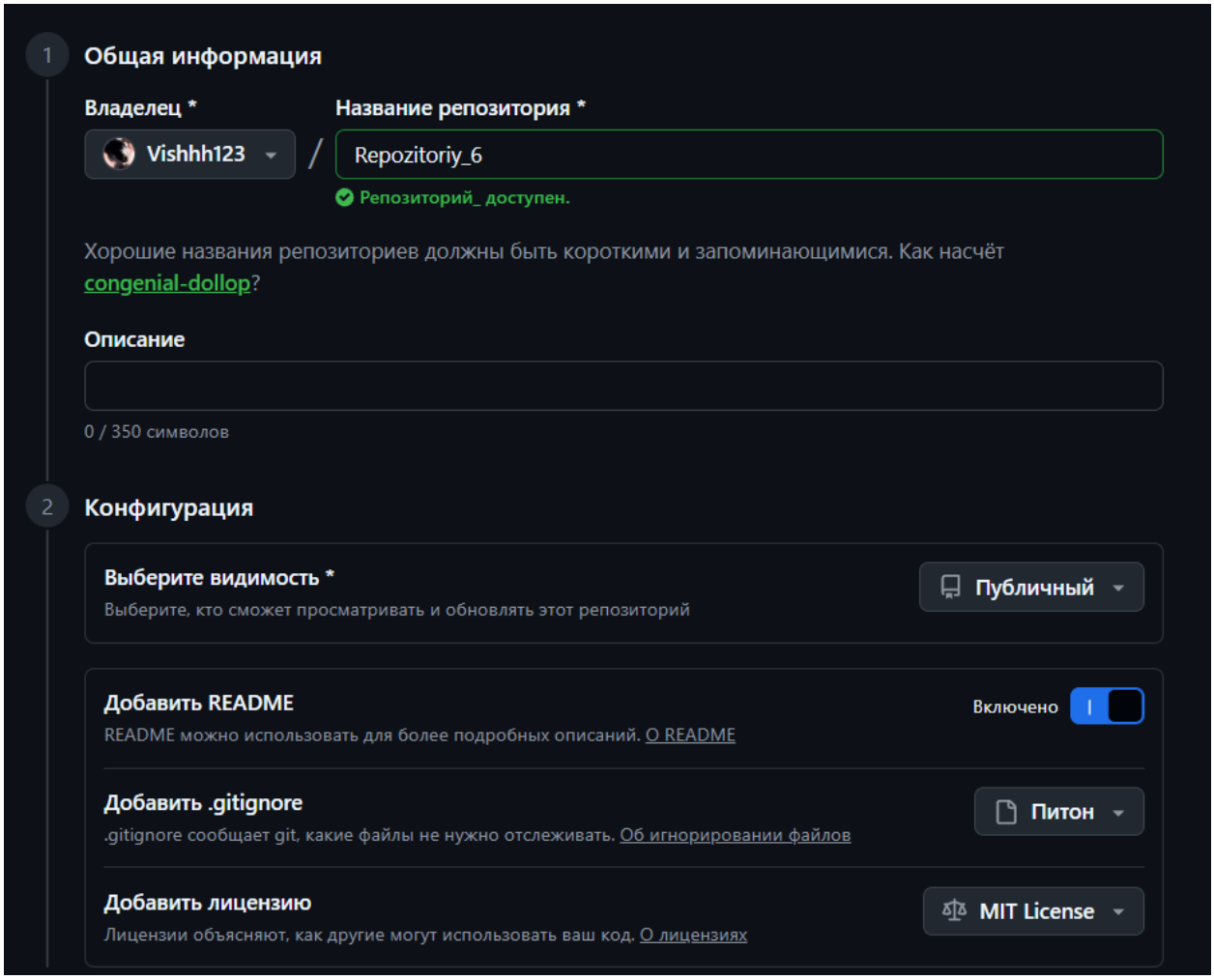

Рис. 1 – создание репозитория.

```
C:\Users\user>git clone https://github.com/Vishhh123/Repozitoriy_6.git
Cloning into 'Repozitoriy_6'...
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 (from 0)
Receiving objects: 100% (5/5), done.

C:\Users\user>cd Repozitoriy_6
```

Рис. 2 – клонирование репозитория, переход к нему.

primer_1.py

```python
#!/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))
```

Run primer_1

```
"C:\Program Files\Python313\python.exe" C:/Users/user/Repozitoriy_6/Примеры/primer_1.py
None
6.0
4.5
```

Рис. 3 – код, результат выполнения примера 1.

primer_2.py

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

import sys
from datetime import date

def get_worker():
    name = input("Фамилия и инициалы: ")
    post = input("Должность: ")
    year = int(input("Год поступления: "))
    return {
        "name": name,
        "post": post,
        "year": year,
    }

def display_workers(staff):
    if staff:
        line = '+-{}-+-{}-+-{}-+-{}-+'.format(
            '-' * 4,
            '-' * 30,
            '-' * 20,
            '-' * 8
        )
        print(line)
        print(
            '| {:^4} | {:^30} | {:^20} |'.format(
                "№",
                "ФИО",
                "Должность",
                "Год"
            )
        )
        print(line)
        for idx, worker in enumerate(staff, 1):
            print(
                '| {:^4} | {:^30} | {:^20} |'.format(
                    idx,
                    worker.get("name", ''),
                    worker.get("post", ''),
                    worker.get("year", 0)
                )
            )
        print(line)
```

Рис. 4 - код к примеру 2.

```
    else:
        print("Список работников пуст.")

def select_worker(staff, period):
    today = date.today()
    result = []
    for employee in staff:
        if today.year - employee.get("year", today.year) >= period:
            result.append(employee)
    return result

def main():
    workers = []
    while True:
        command = input(">>> ").lower()
        if command == "exit":
            break
        elif command == "add":
            worker = get_worker()
            workers.append(worker)
            if len(workers) > 1:
                workers.sort(key=lambda item: item.get("name", ''))
        elif command == "list":
            display_workers(workers)
        elif command.startswith("select "):
            parts = command.split(" ", maxsplit=1)
            period = int(parts[1])
            selected = select_worker(workers, period)
            display_workers(selected)
        elif command == "help":
            print("Список команд:\n")
            print("add - добавить работника")
            print("list - вывести список работников")
            print("select <стаж> - запросить работников со стажем")
            print("help - отоброзить справку")
            print("exit - завершить работу с программой")
        else:
            print(f"Неизвестная команда {command}", file=sys.stderr)
    return 0

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

Рис. 5 - код к примеру 2.

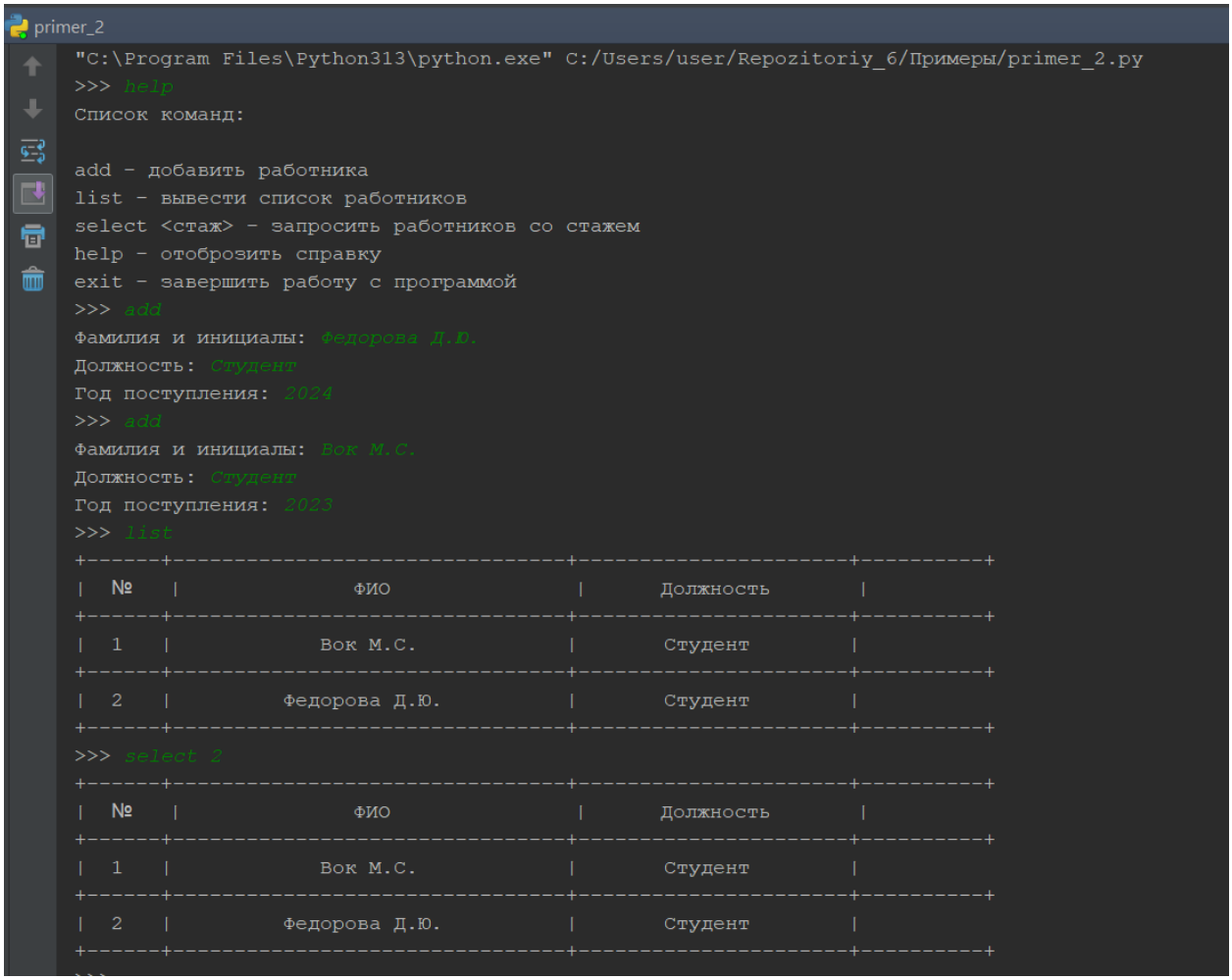

Рис. 6 - результат выполнения примера 2.

7. 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.

Рис. 7 - индивидуальное задание 1.

zadanie_1.py

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

import sys
from datetime import datetime


def get_train():
    destination = input("Пункт назначения: ")
    train_number = input("Номер поезда: ")
    departure_time = datetime.strptime(
        input("Время отправления (в формате HH:MM): "),
        "%H:%M"
    )

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


def display_trains(trains):
    if trains:
        line = '+-{}-+-{}-+-{}-+-{}-+'.format(
            '-' * 4,
            '-' * 25,
            '-' * 15,
            '-' * 20
        )
        print(line)
        print(
            '| {:^4} | {:^25} | {:^15} | {:^20} |'.format(
                "№",
                "Пункт назначения",
                "Номер поезда",
                "Время отправления"
            )
        )
        print(line)

        for idx, train in enumerate(trains, 1):
            time_str = train.get('departure_time').strftime("%H:%M")
```

Рис. 8 - код к инд.заданию 1.

```
            print(
                '| {:>4} | {:<25} | {:<15} | {:<20} |'.format(
                    idx,
                    train.get('destination', ''),
                    train.get('train_number', ''),
                    time_str
                )
            )
            print(line)
    else:
        print("Список поездов пуст.")


def select_trains(trains, destination):
    result = []
    for train in trains:
        if train.get('destination').lower() == destination.lower():
            result.append(train)
    return result


def main():
    trains = []

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

        if command == "exit":
            break

        elif command == "add":
            train = get_train()
            trains.append(train)

            if len(trains) > 1:
                trains.sort(key=lambda item: item.get('departure_time'))

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

        elif command.startswith("select "):
            parts = command.split(" ", maxsplit=1)
            destination = parts[1]
```

Рис. 9 - код к инд.заданию 1.

```
            selected = select_trains(trains, destination)

            if selected:
                display_trains(selected)
            else:
                print("Поездов в пункт '{}' не найдено.".format(destination))

        elif command == "help":
            print("Список команд:\n")
            print("add - добавить поезд")
            print("list - вывести список всех поездов")
            print("select <пункт назначения> - найти поезда по пункту назначения")
            print("help - отобразить справку")
            print("exit - завершить работу с программой")

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

    return 0


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

Рис. 10 - код к инд. заданию 1.

Реализуйте функцию `compare_objects(*args, **kwargs)`, которая сравнивает словари по указанным ключам (`keys` из `kwargs`). Возвращает словарь: `{ключ: [значения]}` для сравниваемых объектов.

Рис. 11 - индивидуальное задание 2.

primer_2.py

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

def compare_objects(*args, **kwargs):
    keys = kwargs.get("keys", [])

    result = {}

    for key in keys:
        values = []
        for obj in args:
            value = obj.get(key)
            values.append(value)
        result[key] = values

    return result


if __name__ == "__main__":
    result = compare_objects(
        {"a": 1, "b": 2},
        {"a": 3, "b": 2},
        keys=["a", "b"]
    )
    print(result)
```

Run: primer_2 primer_2 (1)

```
"C:\Program Files\Python313\python.exe" C:/Users/user/Repozitoriy_6/Задания/primer_2.py
{'a': [1, 3], 'b': [2, 2]}
```

Рис. 12 - код к индивидуальному заданию 2.

Дана последовательность пар:

```
[("root", "A"), ("root", "B"), ("A", "C"), ("B", "D"), ("D", "E")]
```

Создайте функцию `build_tree(pairs, root)`,
которая рекурсивно строит словарь-дерево:

Рис. 13 - индивидуальное задание 3.

```
primer_3.py
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

def build_tree(pairs, root):
    tree = {}

    for parent, child in pairs:
        if parent == root:
            tree[child] = build_tree(pairs, child)

    return tree


pairs = [("root", "A"), ("root", "B"), ("A", "C"), ("B", "D"), ("D", "E")]
result = {"root": build_tree(pairs, "root")}
print(result)

Run: primer_2 primer_3
"C:\Program Files\Python313\python.exe" C:/Users/user/Repozitoriy_6/Задания/primer_3.py
{'root': {'A': {'C': {}}, 'B': {'D': {'E': {}}}}}
```

Рис. 14 - код к индивидуальному заданию 3.

```
C:\Users\user>cd Repozitoriy_6

C:\Users\user\Repozitoriy_6>git add .

C:\Users\user\Repozitoriy_6>git commit -m "создание папок"
[main 5e047d5] создание папок
 1 file changed, 6 insertions(+), 1 deletion(-)

C:\Users\user\Repozitoriy_6>git add .

C:\Users\user\Repozitoriy_6>git commit -m "Примеры"
[main 475ee79] Примеры
 3 files changed, 121 insertions(+)
 create mode 100644 "\320\237\321\200\320\270\320\274\320\265\321\200\321\213/primer_1.py"
 create mode 100644 "\320\237\321\200\320\270\320\274\320\265\321\200\321\213/primer_2.py"
 create mode 100644 "\320\237\321\200\320\270\320\274\320\265\321\200\321\213/primer_3.py"

C:\Users\user\Repozitoriy_6>git add .

C:\Users\user\Repozitoriy_6>git commit -m "задания"
[main 4d8c57d] задания
 3 files changed, 158 insertions(+)
 create mode 100644 "\320\227\320\260\320\264\320\260\320\275\320\270\321\217/primer_2.py"
 create mode 100644 "\320\227\320\260\320\264\320\260\320\275\320\270\321\217/primer_3.py"
 create mode 100644 "\320\227\320\260\320\264\320\260\320\275\320\270\321\217/zadanie_1.py"

C:\Users\user\Repozitoriy_6>git push
Enumerating objects: 17, done.
Counting objects: 100% (17/17), done.
Delta compression using up to 24 threads
Compressing objects: 100% (15/15), done.
Writing objects: 100% (15/15), 3.91 KiB | 668.00 KiB/s, done.
Total 15 (delta 2), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (2/2), done.
To https://github.com/Vishhh123/Repozitoriy_6.git
   074dfb0..4d8c57d  main -> main
```

Рис. 15 – сохранение, отправка на GitHub.

Ответы на контрольные вопросы:

1. Назначение функций в Python
Блоки кода для выполнения конкретных задач. Устраняют дублирование, делают программу модульной.
2. Назначение операторов def и return
def — создаёт функцию. return — возвращает результат или завершает функцию.
3. Локальные и глобальные переменные
Локальные — существуют только внутри функции. Глобальные — видны во всей программе.
4. Вернуть несколько значений из функции
Возврат кортежа: return a, b, c. Можно распаковать: x, y, z = func().
5. Способы передачи значений в функцию
Позиционные (по порядку) и именованные (по имени).
Также *args и **kwargs.
6. Значения аргументов по умолчанию
def greet(name="Гость"): — если аргумент не передан, используется значение по умолчанию.
7. Назначение lambda-выражений
Краткие анонимные функции. Пример: add = lambda x, y: x + y.
8. Документирование кода по PEP257
Использование строк документации (docstring) в тройных кавычках сразу после def.
9. Однострочные и многострочные строки документации
Однострочные — краткое описание. Многострочные — полная документация с параметрами, примерами.
10. Позиционные аргументы
Аргументы, передаваемые в порядке объявления параметров: func(1, 2, 3).
11. Именованные аргументы
Аргументы с указанием имени параметра: func(a=1, b=2, c=3).
12. Оператор *
Распаковывает коллекции или собирает позиционные аргументы в кортеж *args.
13. *args и **kwargs
*args — кортеж всех позиционных аргументов. **kwargs — словарь всех именованных аргументов.
14. Для чего нужна рекурсия
Решение задач, которые можно разбить на одинаковые подзадачи: обход деревьев, факториал, алгоритмы.
15. Что называется базой рекурсии
Условие выхода из рекурсии, которое останавливает цепочку вызовов.
17. Получить текущее значение глубины рекурсии
import sys; sys.getrecursionlimit() — возвращает текущий лимит (~1000).

18. Превышение глубины рекурсии
Вызывается исключение RecursionError — переполнение стека вызовов.
19. Изменить глубину рекурсии
sys.setrecursionlimit(2000) — увеличивает максимальное количество вложенных вызовов.
20. Декоратор lru_cache
Кэширует результаты вызовов функции. Ускоряет повторные вычисления, особенно в рекурсии.
21. Хвостовая рекурсия и оптимизация
Рекурсия, где рекурсивный вызов — последняя операция. Python не оптимизирует хвостовую рекурсию, в отличие от некоторых языков.

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