

**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](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 x
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  def median(*args):
5      if args:
6          values = [float(arg) for arg in args]
7          values.sort()
8          n = len(values)
9          idx = n // 2
10         if n % 2:
11             return values[idx]
12         else:
13             return (values[idx - 1] + values[idx]) / 2
14     else:
15         return None
16
17 if __name__ == '__main__':
18     print(median())
19     print(median(3, 7, 1, 6, 9))
20     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 x
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  import sys
5  from datetime import date
6
7  def get_worker():
8      name = input("Фамилия и инициалы: ")
9      post = input("Должность: ")
10     year = int(input("Год поступления: "))
11     return {
12         "name": name,
13         "post": post,
14         "year": year,
15     }
16
17 def display_workers(staff):
18     if staff:
19         line = '+-{}-+-{}-+-{}-+-{}-+'.format(
20             '-' * 4,
21             '-' * 30,
22             '-' * 20,
23             '-' * 8
24         )
25         print(line)
26         print(
27             '| {:^4} | {:^30} | {:^20} |'.format(
28                 "№",
29                 "ФИО",
30                 "Должность",
31                 "Год"
32             )
33         )
34         print(line)
35         for idx, worker in enumerate(staff, 1):
36             print(
37                 '| {:^4} | {:^30} | {:^20} |'.format(
38                     idx,
39                     worker.get("name", ''),
40                     worker.get("post", ''),
41                     worker.get("year", 0)
42                 )
43             )
44             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 - отобразить 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.

```
primer_2
"C:\Program Files\Python313\python.exe" C:/Users/user/Repozitoriy_6/Примеры/primer_2.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: Федорова Д.Ю.
Должность: Студент
Год поступления: 2024
>>> add
Фамилия и инициалы: Вок М.С.
Должность: Студент
Год поступления: 2023
>>> list
+-----+-----+-----+-----+
| № |                ФИО                | Должность |
+-----+-----+-----+-----+
| 1 |                Вок М.С.            | Студент   |
+-----+-----+-----+-----+
| 2 |                Федорова Д.Ю.       | Студент   |
+-----+-----+-----+-----+
>>> select 2
+-----+-----+-----+-----+
| № |                ФИО                | Должность |
+-----+-----+-----+-----+
| 1 |                Вок М.С.            | Студент   |
+-----+-----+-----+-----+
| 2 |                Федорова Д.Ю.       | Студент   |
+-----+-----+-----+-----+
>>>
```

Рис. 6 - результат выполнения примера 2.

```
def recursion(n):
    if n == 1:
        return 1
    return n + recursion(n-1)

print(recursion(5))

primer_3
"C:\Program Files\Python313\python.exe" C:/Users/user/Repozitoriy_6/Примеры/primer_3.py
15
```

Рис. 7 – код к примеру 3.

- 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. 8 - индивидуальное задание 1.

```

1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  import sys
5  from datetime import datetime
6
7
8  def get_train():
9      destination = input("Пункт назначения: ")
10     train_number = input("Номер поезда: ")
11     departure_time = datetime.strptime(
12         input("Время отправления (в формате HH:MM): "),
13         "%H:%M"
14     )
15
16     return {
17         'destination': destination,
18         'train_number': train_number,
19         'departure_time': departure_time,
20     }
21
22
23 def display_trains(trains):
24     if trains:
25         line = '+-()-+-()-+-()-+-()-+'.format(
26             '-' * 4,
27             '-' * 25,
28             '-' * 15,
29             '-' * 20
30         )
31         print(line)
32         print(
33             '| {:^4} | {:^25} | {:^15} | {:^20} |'.format(
34                 "№",
35                 "Пункт назначения",
36                 "Номер поезда",
37                 "Время отправления"
38             )
39         )
40         print(line)
41
42     for idx, train in enumerate(trains, 1):
43         time_str = train.get('departure_time').strftime("%H:%M")

```

Рис. 9 - 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. 10 - 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. 11 - 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. 12 - индивидуальное задание 2.



```

1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  def compare_objects(*args, **kwargs):
5      keys = kwargs.get("keys", [])
6
7      result = {}
8
9      for key in keys:
10         values = []
11         for obj in args:
12             value = obj.get(key)
13             values.append(value)
14             result[key] = values
15
16     return result
17
18
19 if __name__ == "__main__":
20     result = compare_objects(
21         {"a": 1, "b": 2},
22         {"a": 3, "b": 2},
23         keys=["a", "b"]
24     )
25     print(result)
26
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. 13 - код к индивидуальному заданию 2.



Дана последовательность пар:

```
1 [{"root", "A"}, {"root", "B"}, {"A", "C"}, {"B", "D"}, {"D", "E"}]
```

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

Рис. 14 - индивидуальное задание 3.

```
primer_3.py x
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  def build_tree(pairs, root):
5      tree = {}
6
7      for parent, child in pairs:
8          if parent == root:
9              tree[child] = build_tree(pairs, child)
10
11     return tree
12
13
14     pairs = [{"root", "A"}, {"root", "B"}, {"A", "C"}, {"B", "D"}, {"D", "E"}]
15     result = {"root": build_tree(pairs, "root")}
16     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. 15 - 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. 16 – 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.