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
по дисциплине «Программирование на Python»

Выполнил студент Ромащенко Данил группы ИВТ-б-о-24-1

Подпись студента_____
Работа защищена « »_____20__г.
Проверил Воронкин Р.А. _____
(подпись)

Ставрополь 2025

Тема: «Работа с функциями в языке Python».

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

Ход работы

Вариант 18

Ссылка на репозиторий: https://github.com/l1N322/lab-6.git

1) Был создан общедоступный репозиторий, в котором использована лицензия MIT и язык программирование Python.

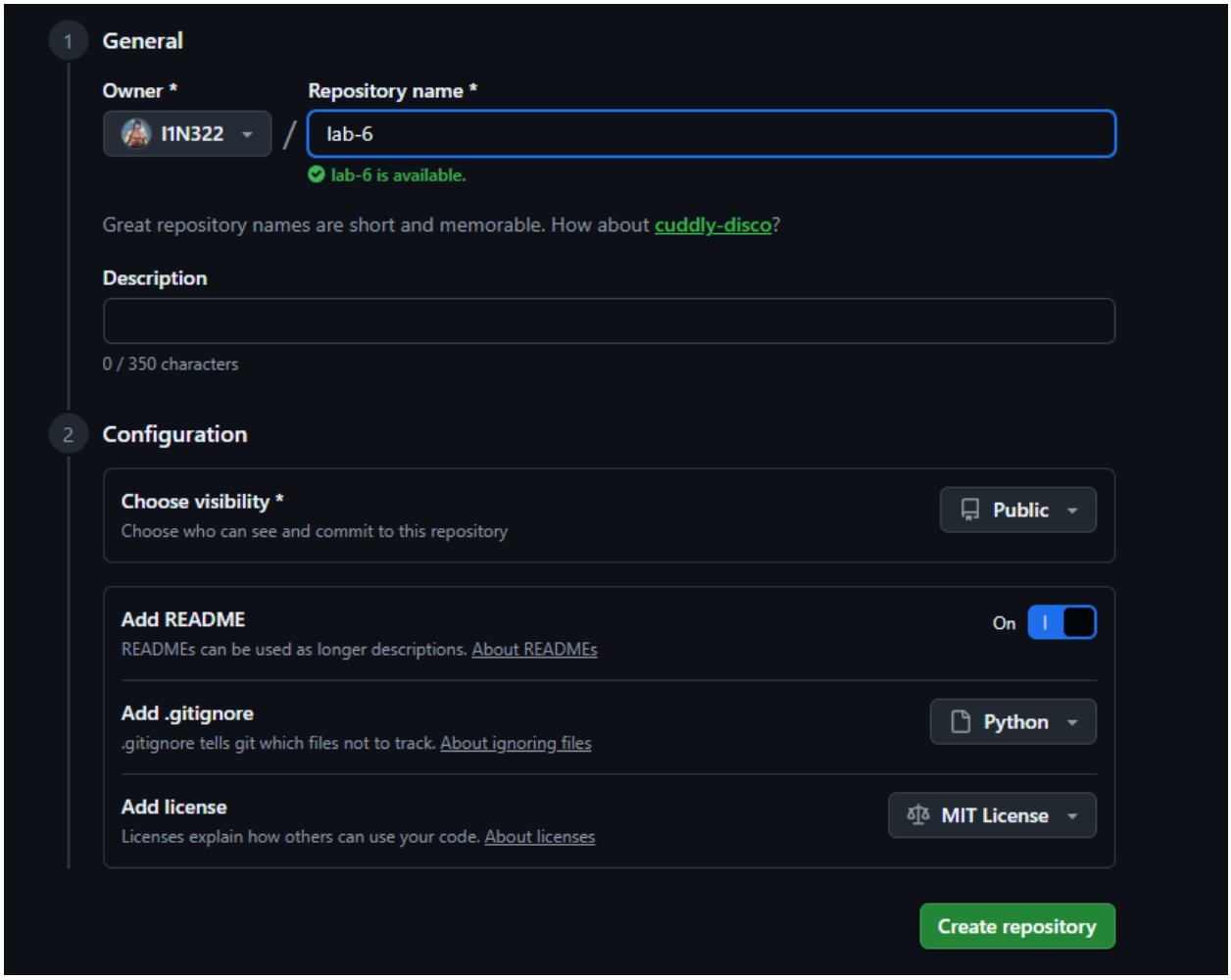

Рисунок 1. Созданный репозиторий

2) Был дополнен файл .gitignore необходимыми правилами для работы с IDE PyCharm.

```
208
209     # Covers JetBrains IDEs: IntelliJ, GoLand, RubyMine, PhpStorm, AppCode, PyCharm, CLion, Android Studio, WebStorm and Rider
210     # Reference: https://intellij-support.jetbrains.com/hc/en-us/articles/206544839
211
212     # User-specific stuff
213     .idea/**/workspace.xml
214     .idea/**/tasks.xml
215     .idea/**/usage.statistics.xml
216     .idea/**/dictionaries
217     .idea/**/shelf
218
219     # AWS User-specific
220     .idea/**/aws.xml
221
222     # Generated files
223     .idea/**/contentModel.xml
224
225     # Sensitive or high-churn files
226     .idea/**/dataSources/
227     .idea/**/dataSources.ids
228     .idea/**/dataSources.local.xml
229     .idea/**/sqlDataSources.xml
230     .idea/**/dynamic.xml
231     .idea/**/uiDesigner.xml
```

Рисунок 2. Добавленные правила в файл .gitignore

3) Было выполнено клонирование созданного репозитория на компьютер.

```
USER@topPC MINGW64 ~
$ git clone https://github.com/l1N322/lab-6.git
Cloning into 'lab-6'...
remote: Enumerating objects: 8, done.
remote: Counting objects: 100% (8/8), done.
remote: Compressing objects: 100% (7/7), done.
remote: Total 8 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (8/8), 5.21 KiB | 5.21 MiB/s, done.
Resolving deltas: 100% (1/1), done.
```

Рисунок 3. Клонирование репозитория

4) Был создан проект PyCharm в папке репозитория.

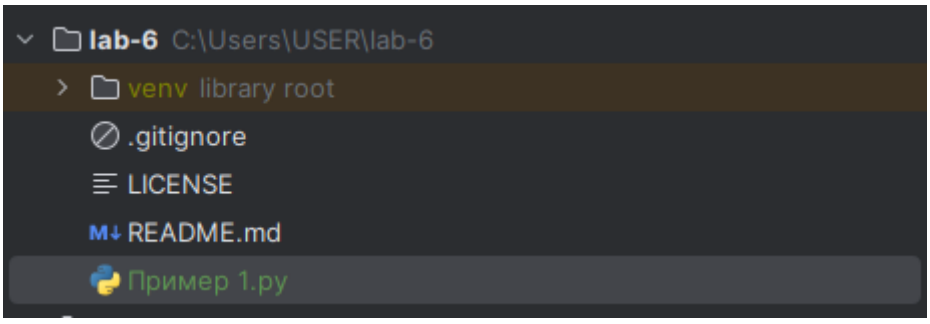

Рисунок 4. Созданный проект

5) Были проработаны примеры лабораторной работы и зафиксированы результаты выполнения каждой из программ при разных исходных данных, вводимых с клавиатуры.

```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))
```

Рисунок 5. Пример 1

```
None
6.0
4.5

Process finished with exit code 0
```

Рисунок 6. Результат выполнения примера 1

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

    return {
        'name': name,
        'post': post,
        'year': year,
    }


def display_workers(staff):
    """
    Отобразить список работников.
    """
    if staff:
        line = '+-{}-+-{}-+-{}-+-{}-+'.format(
            '-' * 4,
            '-' * 30,
            '-' * 20,
            '-' * 8
        )
        print(line)
        print(
            '| {:^4} | {:^30} | {:^20} | {:^8} |'.format(
                "№",
                "Ф.И.О.",
                "Должность",
                "Год"
            )
        )
        print(line)

        for idx, worker in enumerate(staff, 1):
```

Рисунок 7. Пример 2

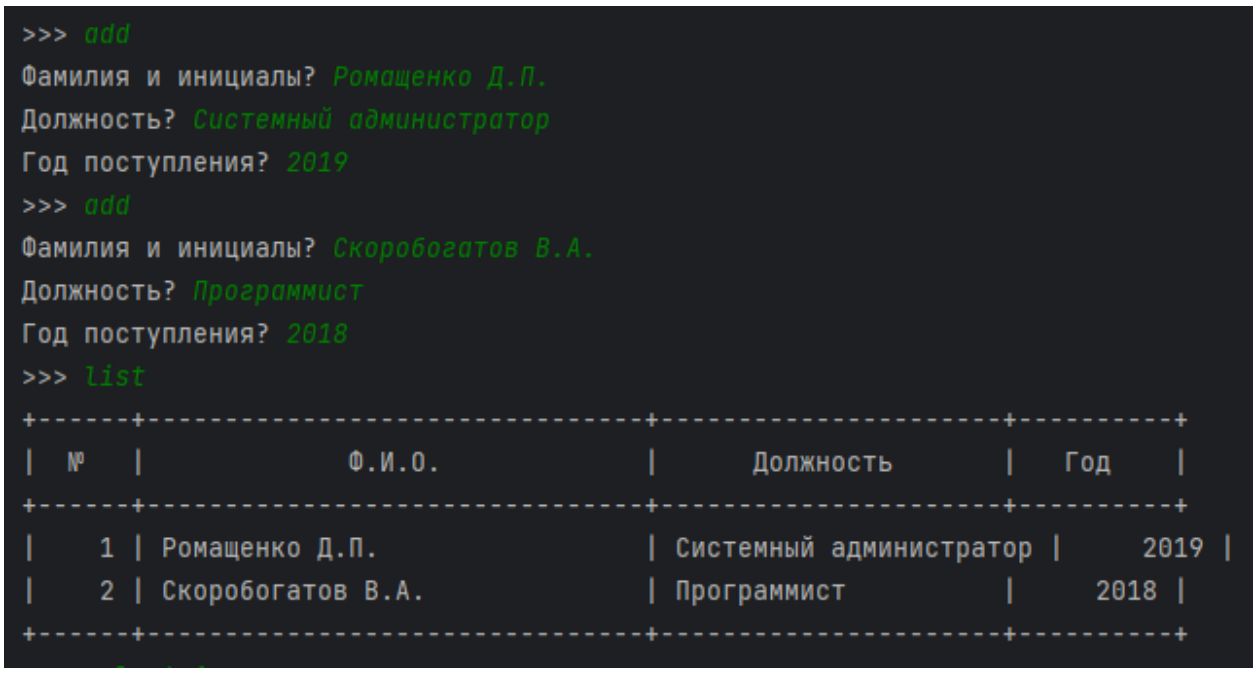

Рисунок 8. Результат выполнения примера 2

6) Были выполнены индивидуальные задания.

7) 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: название товара; название магазина, в котором продается товар; стоимость товара в руб. Написать программу, выполняющую следующие действия: ввод с клавиатуры данных в список, состоящий из словарей заданной структуры; записи должны быть размещены в алфавитном порядке по названиям магазинов; вывод на экран информации о товарах, продающихся в магазине, название которого введено с клавиатуры; если такого магазина нет, выдать на дисплей соответствующее сообщение.

```python
def add_products():
    """
    Добавить новый товар в список
    """
    product = input("Введите название товара: ")
    shop = input("Введите название магазина: ")
    price = float(input("Введите стоимость: "))

    return{
        'товар': product,
        'магазин': shop,
        'цена': price
    }


def list_products(products):
    """
    Отобразить список товаров
    """
    if products:
        print("Все товары:")
        for p in products:
            print(f"{p['магазин']} - {p['товар']} - {p['цена']} руб.")


def search_products(products):
    """
    Найти товары по названию магазина
    """
    search = input("Введите магазин для поиска: ")
    found = [p for p in products if p['магазин'].lower() == search.lower()]

    if found:
        print(f"Товары в магазине '{search}':")
        for p in found:
            print(f"{p['товар']} - {p['цена']} руб.")
    else:
        print(f"Магазин '{search}' не найден.")
```

Рисунок 9. Задание 1

```
>>> help
Список команд:

add - добавить товар;
list - вывести список товаров;
search - вывести информацию о товарах, продающихся в данном магазине;
help - отобразить справку;
exit - завершить работу с программой.
>>> add
Введите название товара: молоко
Введите название магазина: Эконом
Введите стоимость: 55
>>> add
Введите название товара: хлеб
Введите название магазина: Эконом
Введите стоимость: 60
>>> add
Введите название товара: сметана
Введите название магазина: Гастроном
Введите стоимость: 80
>>> list
Все товары:
Гастроном - сметана - 80.0 руб.
Эконом - молоко - 55.0 руб.
Эконом - хлеб - 60.0 руб.
>>> search
Введите магазин для поиска: Эконом
Товары в магазине 'Эконом':
молоко - 55.0 руб.
хлеб - 60.0 руб.
>>> exit
```

Рисунок 10. Результат выполнения задания 1

8) Задание 2: Решить задачу с использованием функций с переменным количеством параметров. Задача: создайте функцию merge_profiles(*profiles, **defaults), которая объединяет словари профилей пользователей. При конфликте ключей приоритет у последнего, а недостающие ключи заполняются значениями из defaults

```python
def merge_profiles(*profiles, **defaults):
    result = {}

    for profile in profiles:
        for key, value in profile.items():
            result[key] = value

    for key, value in defaults.items():
        if key not in result:
            result[key] = value

    return result

if __name__ == "__main__":
    merged = merge_profiles(
        {"name": "Alice", "city": "Paris"},
        {"age": 25},
        country="France"
    )
    print("Тест 1:")
    print(merged)

    merged2 = merge_profiles(
        {"name": "Bob", "age": 30},
        {"name": "Robert", "city": "London"},
        {"age": 35},
        country="UK",
        city="Manchester"
    )
    print("\nТест 2 (проверка приоритетов):")
    print(merged2)

    merged3 = merge_profiles(
        name="John",
        age=40,
        city="New York"
    )
    print("\nТест 3:")
    print(merged3)
```

Рисунок 11. Задание 2

```
Тест 1:
{'name': 'Alice', 'city': 'Paris', 'age': 25, 'country': 'France'}

Тест 2 (проверка приоритетов):
{'name': 'Robert', 'age': 35, 'city': 'London', 'country': 'UK'}

Тест 3:
{'name': 'John', 'age': 40, 'city': 'New York'}
```

Рисунок 12. Результат выполнения задания 2

9) Задание 3: Рекурсивное выравнивание вложенного списка. Задача: создайте функцию flatten(data), которая рекурсивно "расплющивает" список любой вложенности в плоский

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

Danil *
def flatten(data):
    """
    Рекурсивно преобразует вложенные списки, кортежи и множества в плоский список.
    """
    result = []

    for element in data:
        if isinstance(element, (list, tuple, set)):
            result.extend(flatten(element))
        else:
            result.append(element)

    return result


if __name__ == "__main__":
    print(flatten([1, [2, [3, 4], 5]]))
    print(flatten([0, [1, 2, [3, [4], 5], 6]]))
    print(flatten([1, (2, 3), [4, 5]]))
    print(flatten([1, {2, 3}, (4, [5, 6])]))
    print(flatten((1, [2, 3], {4, 5, 6})))
```

Рисунок 13. Задание 3

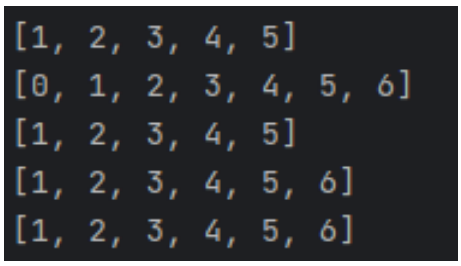

Рисунок 14. Результат выполнения задания 3

10) Были зафиксированы и отправлены сделанные изменения на сервер GitHub.

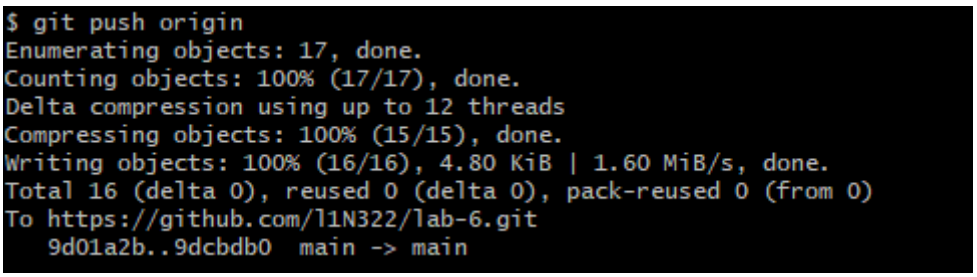

Рисунок 15. Отправка изменений

Ответы на 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__ ру.

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 ТСО 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.