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

Институт цифрового развития
Кафедра инфокоммуникаций

ОТЧЕТ
ПО ПРАКТИЧЕСКОЙ РАБОТЕ №14
дисциплины «Программирование на Python»
Вариант 23

Выполнил:
Мотовилов Вадим Борисович
2 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и
вычислительная техника»,
направленность (профиль)
«Информатика и вычислительная
техника», очная форма обучения

_____
(подпись)

Руководитель практики:
Воронкин Роман Александрович
_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2023 г.

Порядок выполнения работы:

1. Создал репозиторий и скопировал его

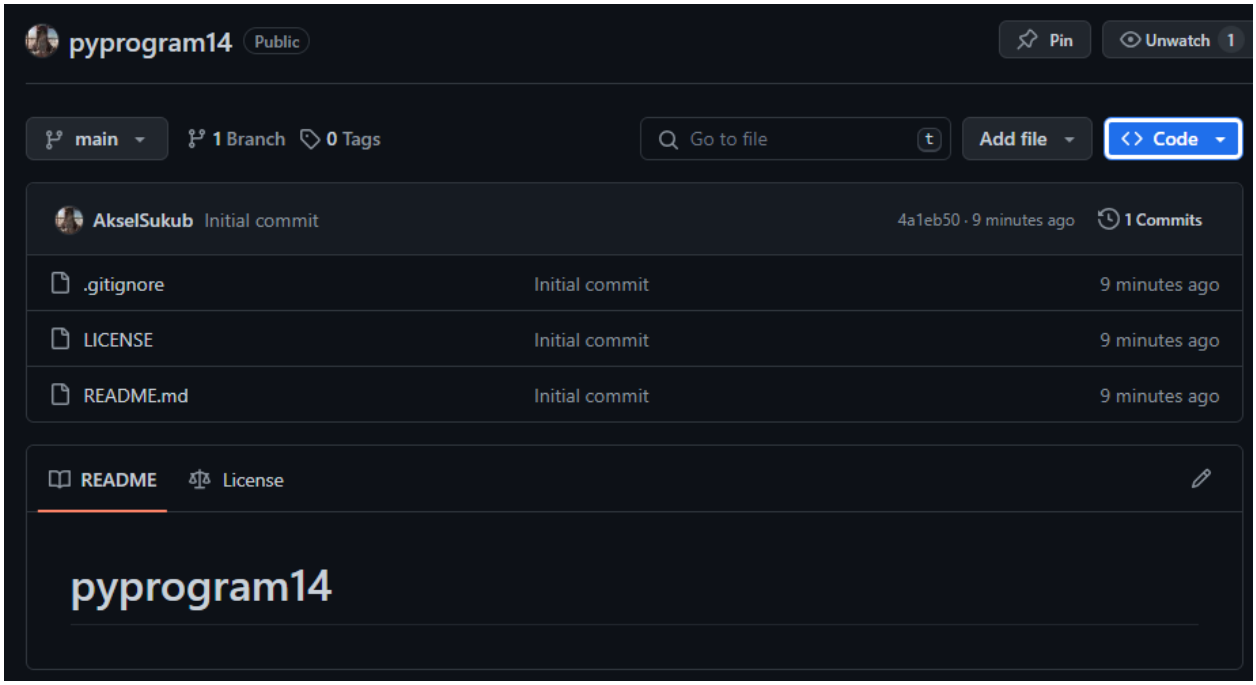

Рисунок 1. Созданный репозиторий

2. Изменил файл .gitignore и README.rm и добавил git flow

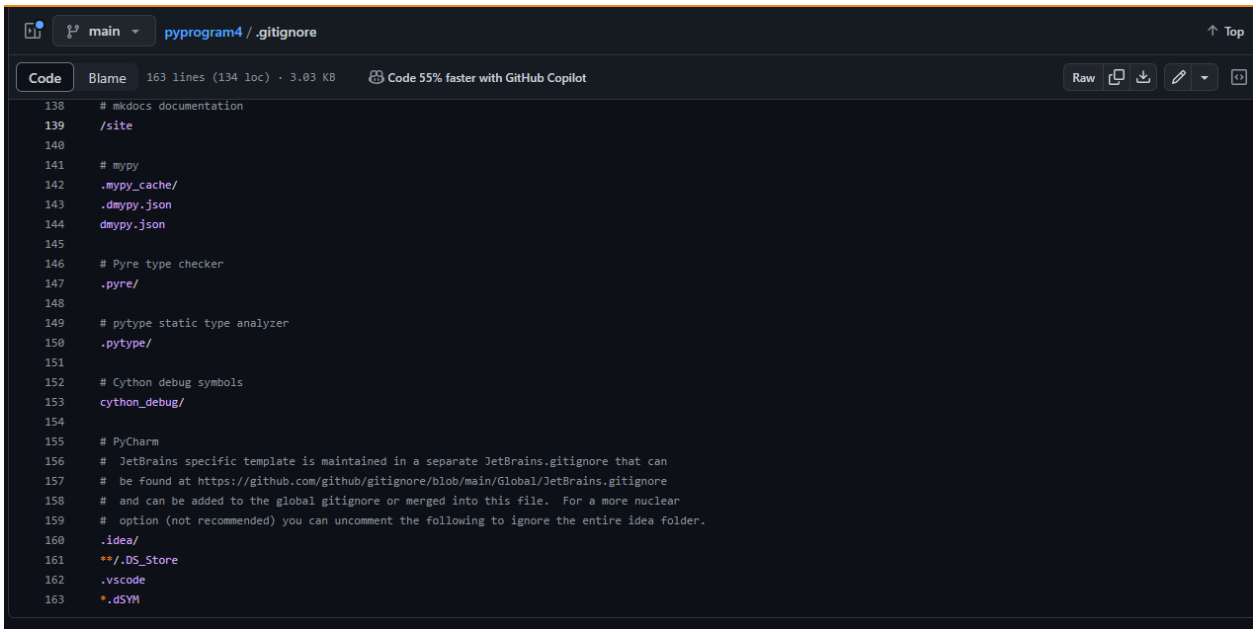

Рисунок 2. Измененный файл .gitignore

3. Выполнил задания

Используя замыкания функций, объявите внутреннюю функцию, которая преобразует строку из списка целых чисел, записанных через пробел, либо в список, либо в кортеж. Тип коллекции определяется параметром `type` внешней функции. Если `type = 'list'`, то используется список, иначе – кортеж. Далее, на вход программы поступает две строки: первая – это значение для параметра type; вторая – список целых чисел, записанных через

пробел. С помощью реализованного замыкания преобразовать эту строку в соответствующую коллекцию. Результат работы замыкания выведите на экран.

zadanie1.py > ...

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

def collection_converter(type):
    def convert_to_collection(numbers):
        if type == 'list':
            collection = list(map(int, numbers.split()))
        else:
            collection = tuple(map(int, numbers.split()))
        return collection

    return convert_to_collection


type_input = input("Введите тип коллекции ('list' или 'tuple'): ")
numbers_input = input("Введите список целых чисел, записанных через пробел: ")

converter = collection_converter(type_input)
result = converter(numbers_input)
print(result)

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\1\pyprogram14\prog> & "C:/Program Files/Python311/python.exe" c:/Users/1/pyprogram1
Введите тип коллекции ('list' или 'tuple'): list
Введите список целых чисел, записанных через пробел: 2 6 7 8
[2, 6, 7, 8]
PS C:\Users\1\pyprogram14\prog> & "C:/Program Files/Python311/python.exe" c:/Users/1/pyprogram1
Введите тип коллекции ('list' или 'tuple'): tuple
Введите список целых чисел, записанных через пробел: 3 5 6 9
(3, 5, 6, 9)
PS C:\Users\1\pyprogram14\prog>
```

Рисунок 3 . Выполнение 1 задания

Ответы на контрольные вопросы:

1. **Замыкание (Closure)** — это функция, которая имеет доступ к переменным из своей внешней области видимости, даже после того, как внешняя функция завершила свою работу. Замыкания позволяют "запомнить" и использовать значения переменных из внешней области видимости.

2. **Реализация замыканий в Python**: В Python замыкания реализованы с помощью вложенных функций. Внутренняя функция имеет доступ к переменным внешней функции, даже после того, как внешняя функция завершила свою работу.

3. **Область видимости Local (Локальная область видимости)**: Это область внутри функции, где объявлены переменные. Эти переменные доступны только внутри этой функции.

4. **Область видимости Enclosing (Охватывающая область видимости)**: Это область между локальной и глобальной областями видимости. Она применима для вложенных функций и охватывает область видимости внешней функции.

5. **Область видимости Global (Глобальная область видимости)**: Это область вне всех функций. Переменные, объявленные здесь, доступны в любом месте в коде.

6. **Область видимости Built-in (Встроенная область видимости)**: Это специальная область, где Python хранит встроенные функции и исключения, доступные в любом месте в коде.

7. **Использование замыканий в Python**: Замыкания в Python могут быть использованы для сохранения состояния между вызовами функций, для создания функций данных, для реализации декораторов и для других задач, где необходимо сохранить состояние или данные между вызовами функций.

8. **Использование замыканий для построения иерархических данных**: Замыкания могут использоваться для создания структур данных с вложенностью, где каждый уровень вложенности может иметь свое собственное состояние, сохраняемое между вызовами. Это может быть

полезно, например, при создании деревьев или графов, где каждый узел является замыканием, хранящим свое собственное состояние и имеющим доступ к состоянию своего родителя.