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 №11
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

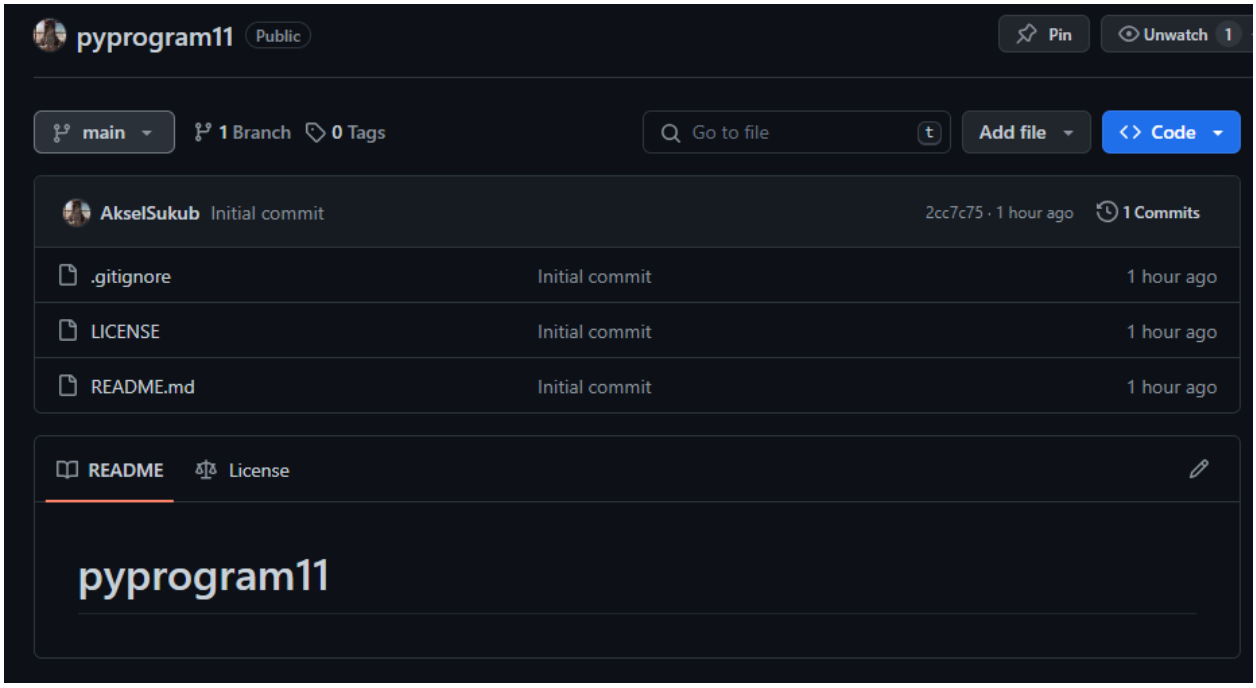

Рисунок 1. Созданный репозиторий

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

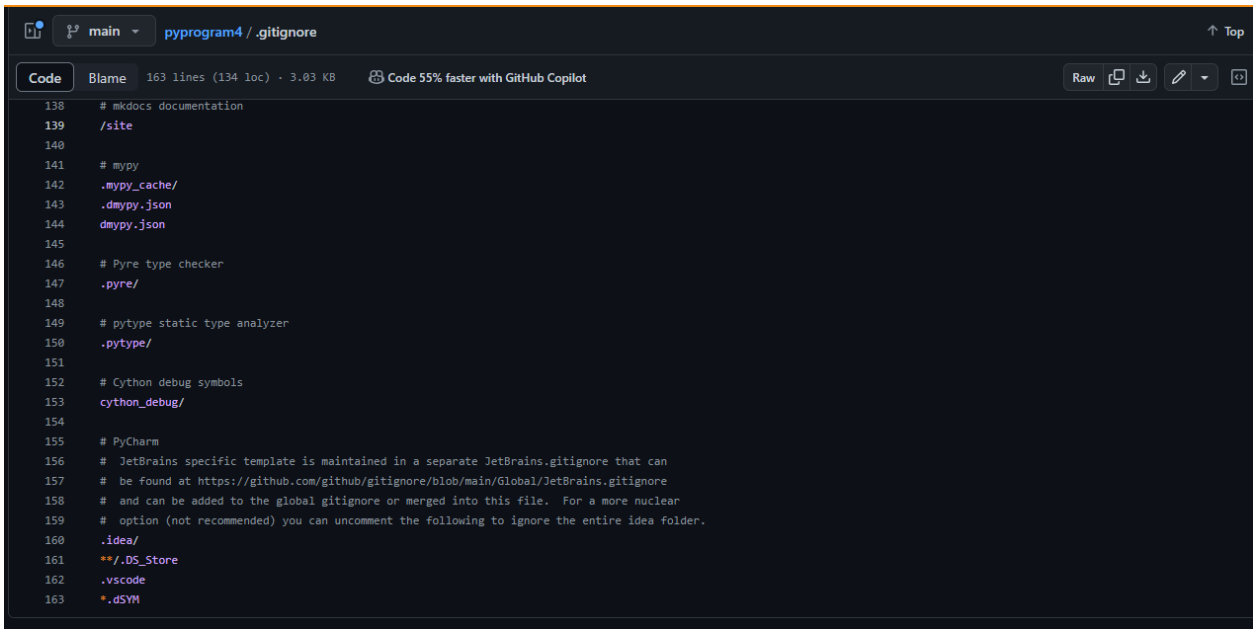

Рисунок 2. Измененный файл .gitignore

3. Выполнил задания

Решить следующую задачу: основная ветка программы, не считая заголовков функций, состоит из двух строки кода. Это вызов функции *test()* и инструкции `if __name__ == '__main__'`. В ней запрашивается на ввод целое число. Если оно положительное, то вызывается функция *positive()*, тело которой содержит команду вывода на экран слова "Положительное". Если число отрицательное, то вызывается функция *negative()*, ее тело содержит выражение вывода на экран слова "Отрицательное".

Понятно, что вызов *test()* должен следовать после определения функций. Однако имеет ли значение порядок определения самих функций? То есть должны ли определения *positive()* и *negative()* предшествовать *test()* или могут следовать после него? Проверьте вашу гипотезу, поменяв объявления функций местами. Попробуйте объяснить результат.

1. В Python порядок определения функций имеет значение. В данном случае, если определения функций positive() и negative() следуют после вызова функции test(), то возникнет ошибка "NameError: name 'positive' (или 'negative') is not defined". Это связано с тем, что интерпретатор Python выполняет код последовательно сверху вниз, и для успешного вызова функций они должны быть определены до момента их вызова.

Таким образом, правильный порядок в данном случае будет следующим:

Рисунок 3 . Выполнение 1 задания

В этом случае код будет работать корректно: пользователю будет предложено ввести целое число, и в зависимости от его значения будет вызвана соответствующая функция positive() или negative().

2. Решите следующую задачу: в основной ветке программы вызывается функция cylinder(), которая вычисляет площадь цилиндра. В теле cylinder() определена функция circle(), вычисляющая площадь круга по формуле πr^2. В теле cylinder() у пользователя спрашивается, хочет ли он получить только площадь боковой поверхности цилиндра, которая вычисляется по формуле $2\pi rh$, или полную площадь цилиндра. В последнем случае к площади боковой поверхности цилиндра должен добавляться удвоенный результат вычислений функции circle().

Рисунок 4 . Выполнение 2 задания

Решите следующую задачу: напишите функцию, которая считывает с клавиатуры числа и перемножает их до тех пор, пока не будет введен 0. Функция должна возвращать полученное произведение. Вызовите функцию и выведите на экран результат ее работы.

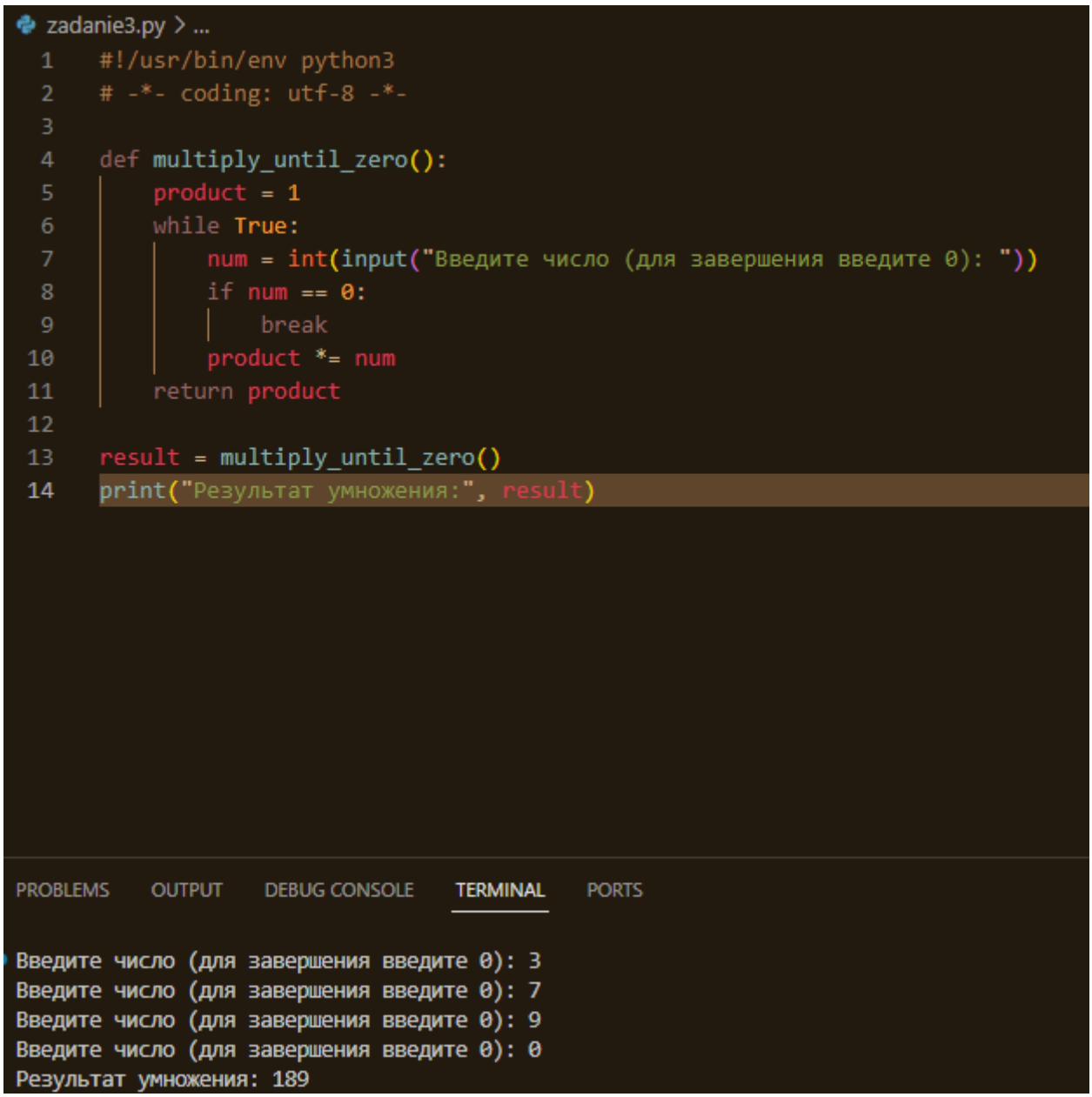

Рисунок 5 . Выполнение 3 задания

Решите следующую задачу: напишите программу, в которой определены следующие четыре функции:

1. Функция *get_input()* не имеет параметров, запрашивает ввод с клавиатуры и возвращает в основную программу полученную строку.
2. Функция *test_input()* имеет один параметр. В теле она проверяет, можно ли переданное ей значение преобразовать к целому числу. Если можно, возвращает логическое *True*. Если нельзя – *False*.

3. Функция *str_to_int()* имеет один параметр. В теле преобразовывает переданное значение к целочисленному типу. Возвращает полученное число.
4. Функция *print_int()* имеет один параметр. Она выводит переданное значение на экран и ничего не возвращает.

В основной ветке программы вызовите первую функцию. То, что она вернула, передайте во вторую функцию. Если вторая функция вернула *True*, то те же данные (из первой функции) передайте в третью функцию, а возвращенное третьей функцией значение – в четвертую.

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

def get_input():
    return input("Введите значение: ")

def test_input(value):
    try:
        int(value)
        return True
    except ValueError:
        return False

def str_to_int(value):
    return int(value)

def print_int(value):
    print(value)

# Основная программа
input_value = get_input()

if test_input(input_value):
    integer_value = str_to_int(input_value)
    print_int(integer_value)
else:
    print("Невозможно преобразовать в целое число.")
```

Рисунок 6 . Выполнение 4 задания

Ответы на контрольные вопросы:

1. Назначение функций в языке программирования Python:

Структурирование кода: Функции позволяют разделить код на более мелкие и понятные части, что упрощает чтение, понимание и поддержку программы.

Повторное использование: Функции могут быть вызваны из разных частей программы или из других программ, что позволяет повторно использовать код и избегать дублирования.

Абстракция: Функции позволяют создавать абстрактные уровни, скрывающие детали реализации и позволяющие сосредоточиться на более высоком уровне абстракции.

Улучшение модульности: Функции помогают организовать код в модули и пакеты, упрощая разработку и поддержку больших программных проектов.

2. Оператор def используется для определения функции в Python. Он указывает, что следующий блок кода является определением функции. Синтаксис оператора def выглядит следующим образом: def function_name(parameters):. Здесь function_name - имя функции, parameters - список параметров функции.

Оператор return используется внутри функции для возврата значения из функции обратно в вызывающую программу. Он указывает, какое значение должно быть возвращено из функции. Если оператор return не указан, функция возвращает None по умолчанию.

3. Локальные переменные в Python определены внутри функции и видны только внутри этой функции. Они существуют только во время выполнения функции и уничтожаются после завершения функции. Локальные переменные используются для временного хранения данных, которые нужны только внутри функции.

Глобальные переменные в Python определены за пределами функций и видны во всей программе. Они могут быть доступны из любой функции в

программе. Глобальные переменные используются для хранения данных, которые должны быть доступны в разных частях программы.

4. В Python можно вернуть несколько значений из функции, используя кортеж, список или словарь. Например:

С использованием кортежа: return value1, value2, ... Значения будут возвращены в виде кортежа.

С использованием списка: return [value1, value2, ...]. Значения будут возвращены в виде списка.

С использованием словаря: return {'key1': value1, 'key2': value2, ...}. Значения будут возвращены в виде словаря.

5. Существуют различные способы передачи значений в функцию в Python:

Позиционные аргументы: Значения передаются в функцию в порядке, соответствующем порядку параметров.

Именованные аргументы: Значения передаются в функцию с указанием имени параметра, в формате parameter_name=value.

Аргументы по умолчанию: Параметры функции могут иметь значения по умолчанию. Если значение не передано при вызове функции, будет использовано значение по умолчанию.

Передача списка или кортежа: С помощью оператора * можно передать список или кортеж элементов в функцию как отдельные аргументы.

Передача словаря: С помощью оператора ** можно передать словарь в функцию как именованные аргументы.

6. Значения аргументов функции по умолчанию могут быть заданы при определении функции. Синтаксис для задания значения по умолчанию выглядит следующим образом: def function_name(parameter=value):. Здесь parameter - имя параметра, value - значение по умолчанию.

7. Lambda-выражения в языке Python представляют собой анонимные функции, которые могут быть определены в одной строке кода. Они часто используются в качестве аргументов функций или вместо

определения отдельной функции. Lambda-выражения могут принимать любое количество аргументов, но могут содержать только одно выражение. Они обычно используются для создания простых функций, которые выполняют небольшие операции.

8. Документирование кода согласно PEP257, который является стандартом для документирования кода в Python, осуществляется с помощью строк документации (docstrings). Строка документации - это строка, которая идет сразу после определения функции, класса или модуля и содержит описание и документацию по использованию.

Для документирования кода согласно PEP257 используются следующие правила:

Строка документации должна быть заключена в тройные кавычки (""").

Строка документации должна содержать описание функции, класса или модуля, а также описание параметров, возвращаемых значений и возможных исключений.

Строка документации должна быть расположена сразу после определения функции, класса или модуля.

Можно использовать специальные теги, такие как Args:, Returns:, Raises:, для явного указания параметров, возвращаемых значений и исключений.

9. Однострочные и многострочные формы строк документации в Python используются для документирования кода:

Однострочная форма: Строка документации помещается в одну строку сразу после определения функции, класса или модуля. Например:

```
def my_function():
"""Описание функции"""
# Код функции
```

Многострочная форма: Строка документации может занимать несколько строк и должна быть заключена в тройные кавычки (""").

Многострочная форма позволяет включать более подробное описание, параметры, возвращаемые значения и исключения. Например:

```
def my_function():
"""
Описание функции.

Args:
param1: Описание параметра 1.
param2: Описание параметра 2.

Returns:
Описание возвращаемого значения.

Raises:
ValueError
```