

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 ЛАБОРАТОРНОЙ РАБОТЕ №11**  
**дисциплины «Основы программной инженерии»**

Выполнила:  
Панюкова Ксения Юрьевна  
2 курс, группа ПИЖ-б-о-22-1,  
09.03.04 «Программная инженерия»,  
направленность (профиль) «Разработка  
и сопровождение программного  
обеспечения», очная форма обучения

---

(подпись)

Руководитель практики:  
Воронкин Р. А., доцент кафедры  
инфокоммуникаций

---

(подпись)

Отчет защищен с оценкой \_\_\_\_\_ Дата защиты \_\_\_\_\_

Ставрополь, 2023 г.

# Ход работы

## 1. Я изучила теоретический материал работы

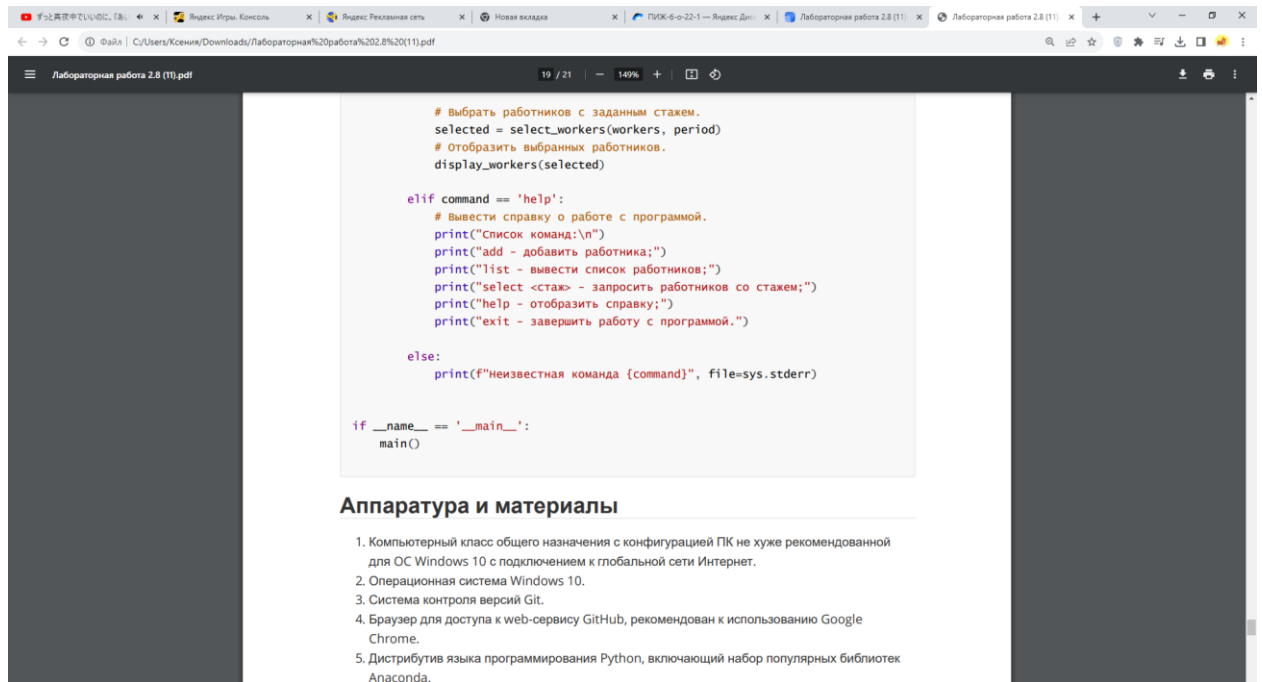


Рисунок 1.1 – Изучение материала для лабораторной работы

2. Создала общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT и язык программирования Python

## Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk (\*).

Owner \*

MrFinichек

Repository name \*

Python>Javascript

✓ Your new repository will be created as Python-Javascript.

The repository name can only contain ASCII letters, digits, and the characters -, ., and \_.

Great repository names are short and memorable. Need inspiration? How about [silver-journey](#) ?

Description (optional)

☒ Public

Anyone on the internet can see this repository. You choose who can commit.

☐ Private

You choose who can see and commit to this repository.

Initialize this repository with:

☐ Add a README file

This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore

.gitignore template: Python

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license

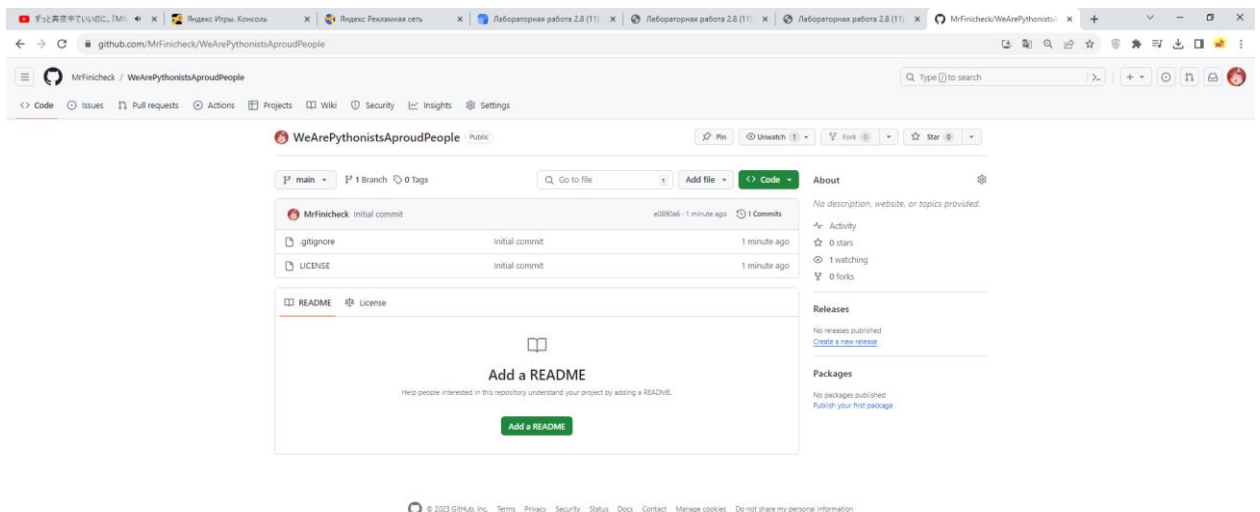
License: MIT License

A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

ⓘ You are creating a public repository in your personal account.

Create repository

## Рисунок 2.1 – Настройка репозитория



## Рисунок 2.2 – Готовый репозиторий

### 3. Выполняю клонирование созданного репозитория

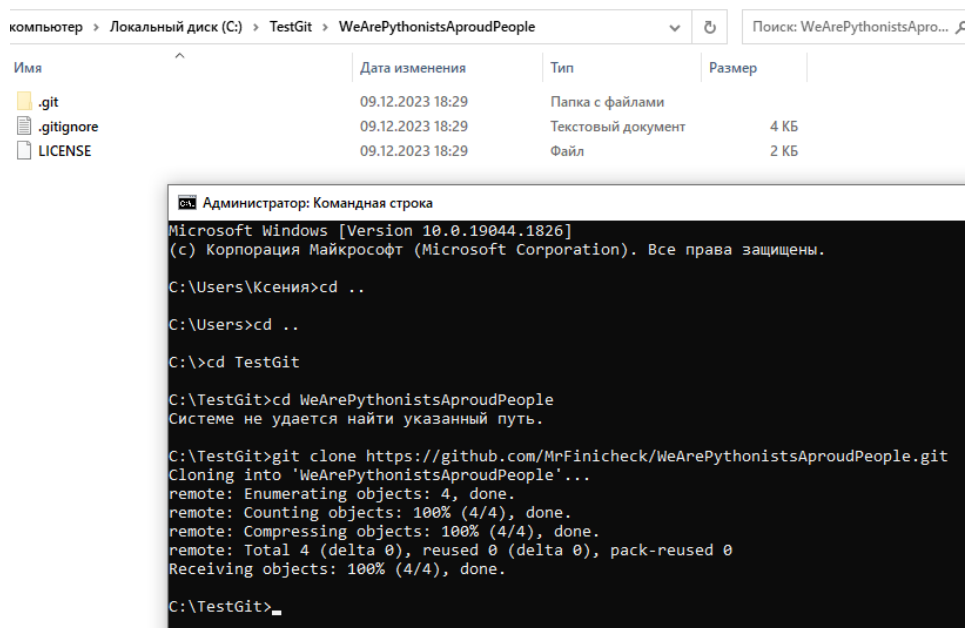


Рисунок 3.1 – Клонирование репозитория на локальный диск

#### 4. Дополнила файл .gitignore необходимыми правилами для работы с IDE PyCharm

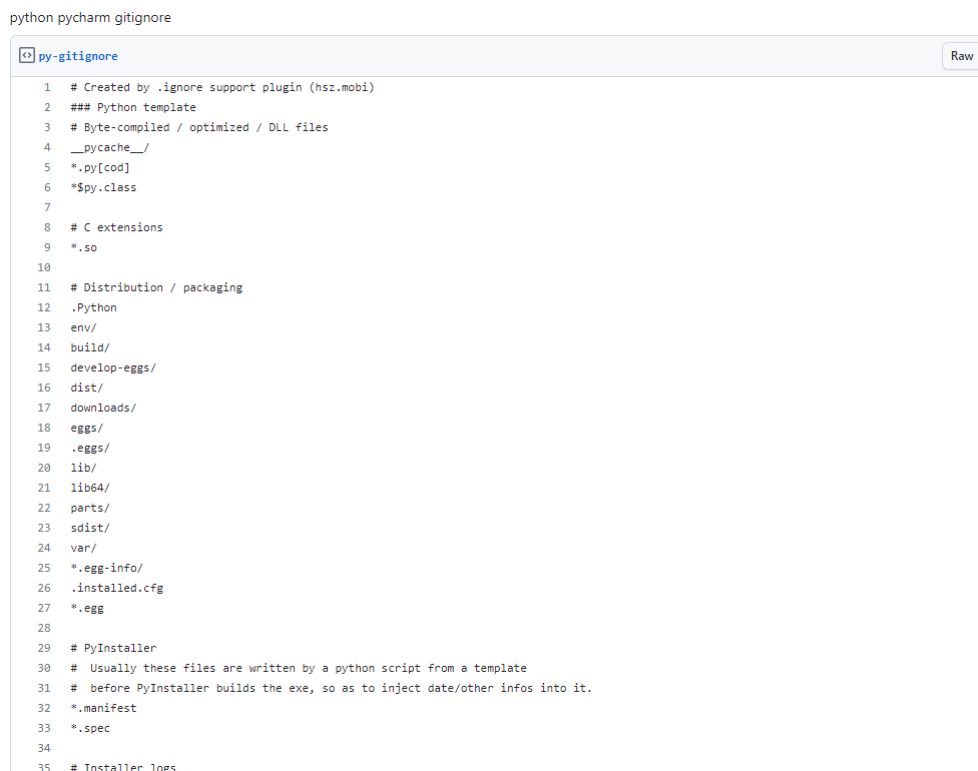


Рисунок 4.1 – .gitignore для IDE PyCharm

5. Организовала свой репозиторий в соответствии с моделью ветвления git-flow

```
C:\TestGit\WeArePythonistsAroudPeople>git branch develop

C:\TestGit\WeArePythonistsAroudPeople>git checkout develop
Switched to branch 'develop'

C:\TestGit\WeArePythonistsAroudPeople>git push -u origin develop
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'develop' on GitHub by visiting:
remote:   https://github.com/MrFinichck/WeArePythonistsAroudPeople/pull/new/develop
remote:
To https://github.com/MrFinichck/WeArePythonistsAroudPeople.git
 * [new branch]      develop -> develop
branch 'develop' set up to track 'origin/develop'.
```

Рисунок 5.1 – Создание ветки develop от ветки main

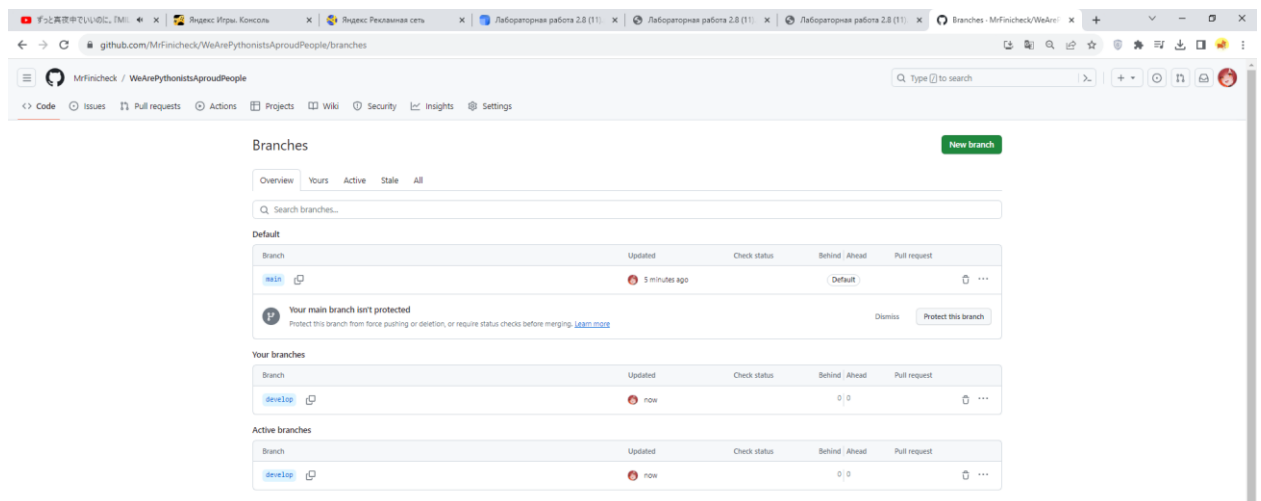


Рисунок 5.2 – Ветка develop на GitHub

6. Создала проект PyCharm в папке репозитория

Имя	Дата изменения	Тип	Размер
.git	09.12.2023 18:32	Папка с файлами	
PyCharm	09.12.2023 18:32	Папка с файлами	
.gitignore	09.12.2023 18:29	Текстовый документ	4 КБ
LICENSE	09.12.2023 18:29	Файл	2 КБ

Рисунок 6.1 – Репозиторий с проектом PyCharm

7. Проработала примеры лабораторной работы. Создала для каждого примера отдельный модуль языка Python. Зафиксировала изменения в репозитории.

```

1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  import sys
5  from datetime import date
6
7
8  def get_worker():
9      """
10     Запросить данные о работнике.
11     """
12     name = input("Фамилия и инициалы? ")
13     post = input("Должность? ")
14     year = int(input("Год поступления? "))
15
16     # Создать словарь.
17     return {
18         'name': name,
19         'post': post,
20         'year': year,
21     }
22
23
24 def display_workers(staff):
25     """
26     Отобразить список работников.
27     """
28     # Проверить, что список работников не пуст.
29     if staff:
30         # Заголовок таблицы.
31         line = '+--{}+--{}+--{}+--{}+'.format(
32             *args: '-' * 4,
33             '-' * 30,
34             '-' * 20,
35             '-' * 8
36         )
37         print(line)
38         print(
39             '| {:^4} | {:^30} | {:^20} | {:^8} |'.format(
40                 *args: "№",
41                 "Ф.И.О.",

```

Рисунок 7.1 – Проработка примера 1

```

C:\TestGit\WeArePythonistsA ProudPeople>git add PyCharm
C:\TestGit\WeArePythonistsA ProudPeople>git commit -m"adding example
[develop c2ce190] adding example
1 file changed, 131 insertions(+)
create mode 100644 PyCharm/examples/EXAmple.py

```

Рисунок 7.5 – Фиксирование изменений в репозитории

8. Решила 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".

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

def test():
    # Ввести целое число.
    number = int(input("Введите целое число: "))

    # Определить знак числа.
    if number > 0:
        positive()
    elif number < 0:
        negative()

def positive():
    print("Положительное")

def negative():
    print("Отрицательное")

if __name__ == '__main__':
    test()
```

Рисунок 8.1 – Код программы Task8.py в IDE PyCharm

## 9. Зафиксировала сделанные изменения в репозитории

```
C:\TestGit\WeArePythonistsA Proud People>git add PyCharm

C:\TestGit\WeArePythonistsA Proud People>git commit -m"add task 8"
[develop 1db159e] add task 8
4 files changed, 27 insertions(+)
create mode 100644 PyCharm/Tasks/TASK10.py
create mode 100644 PyCharm/Tasks/TASK12.py
create mode 100644 PyCharm/Tasks/TASK14.py
create mode 100644 PyCharm/Tasks/TASK8.py
```

Рисунок 9.1 – Коммит файлов в репозитории git

10. Решила следующую задачу: в основной ветке программы вызывается функция `cylinder()`, которая вычисляет площадь цилиндра. В теле `cylinder()` определена функция `circle()`, вычисляющая площадь круга по формуле  $S = \pi r^2$ . В теле `cylinder()` у пользователя спрашивается, хочет ли он получить только площадь боковой поверхности цилиндра, которая вычисляется по формуле  $S_{\text{бок}} = 2\pi r h$ , или полную площадь цилиндра. В последнем случае к площади

боковой поверхности цилиндра должен добавляться удвоенный результат вычислений функции `circle()`.

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

import math

def cylinder():
    # Определить функцию для нахождения площади круга.
    def circle(radius):
        return math.pi * radius ** 2

    # Ввести радиус и высоту цилиндра.
    radius = float(input("Введите радиус цилиндра: "))
    height = float(input("Введите высоту цилиндра: "))

    # Посчитать площади боковой и всей поверхностей.
    lateral_surface_area = 2 * math.pi * radius * height
    total_surface_area = (lateral_surface_area +
                          2 * circle(radius))

    # Узнать хочет ли получить пользователь всю площадь.
    solution = input("Хотите получить только площадь не "
                    "только боковой поверхности, но и "
                    "цилиндра? (Да/Нет): ")

    if solution.lower() == "да":
        print(f"Площадь боковой поверхности цилиндра: "
              f"{lateral_surface_area}")
    else:
```

Рисунок 10.1 – Код программы Task10.py в IDE PyCharm

11. Зафиксировала сделанные изменения в репозитории.

```
C:\TestGit\WeArePythonistsAroudPeople>git add PyCharm

C:\TestGit\WeArePythonistsAroudPeople>git commit -m"add task 10
[develop d6afb5d] add task 10
 1 file changed, 41 insertions(+)
```

Рисунок 11.1 – Коммит файлов в репозитории git

12. Решила 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.



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

def multiply():
    product = 1
    number = int(input("Введите число: "))

    while number != 0:
        product *= number
        number = int(input("Введите число: "))

    print(product)

if __name__ == '__main__':
    multiply()
```

Рисунок 12.1 – Код программы Task12.py в IDE PyCharm

13. Зафиксировала сделанные изменения в репозитории.

```
C:\TestGit\WeArePythonistsA Proud People>git add PyCharm
C:\TestGit\WeArePythonistsA Proud People>git commit -m"add task 12"
[develop 79a9273] add task 12
 2 files changed, 22 insertions(+), 1 deletion(-)
```

Рисунок 13.1 – Коммит файлов в репозитории git

14. Решила 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.

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

# Возвращаем строку.
def get_input():
    line = input("Введите любую строку: ")
    return line

# Проверяем является ли значение числом.
def test_input(variable):
    try:
        int(variable)
        return True
    except ValueError:
        return False

# Преобразовываем переданное значение к целочисленному типу.
def str_to_int(number):
    string_number = int(number)
    return string_number

# Выводим переданное значение на экран.
def print_int(value):
    print(value)

if __name__ == '__main__':
    data = get_input()

    if test_input(data):
        return_value = str_to_int(data)
        print_int(return_value)
    else:
        print("Эта строка не является числом.")
```

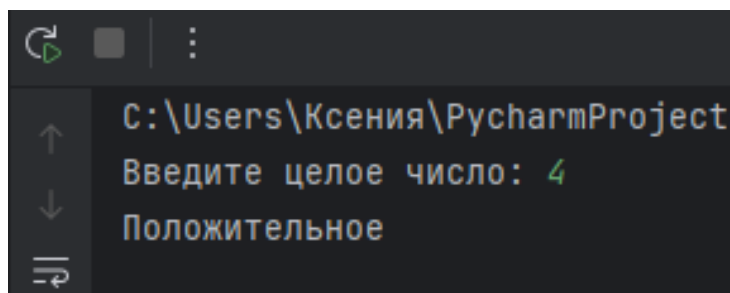
Рисунок 14.1 – Код программы Task14.py в IDE PyCharm

15. Зафиксировала сделанные изменения в репозитории.

```
C:\TestGit\WeArePythonistsAproudPeople>git add PyCharm
C:\TestGit\WeArePythonistsAproudPeople>git commit -m"add task 14"
[develop 7931257] add task 14
2 files changed, 42 insertions(+), 1 deletion(-)
```

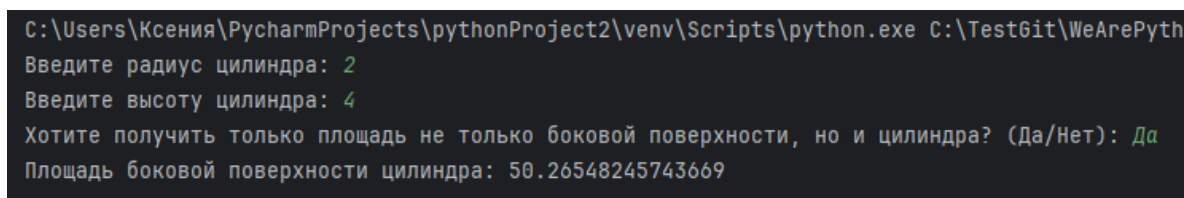
Рисунок 15.1 – Коммит файлов в репозитории git

16. Привела в отчете скриншоты результатов выполнения примера при различных исходных данных, вводимых с клавиатуры.



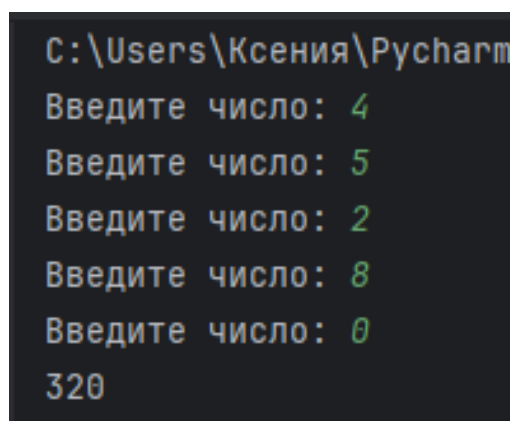
```
C:\Users\Ксения\PycharmProjects
Введите целое число: 4
Положительное
```

Рисунок 16.1 – Результат программы Task8.py



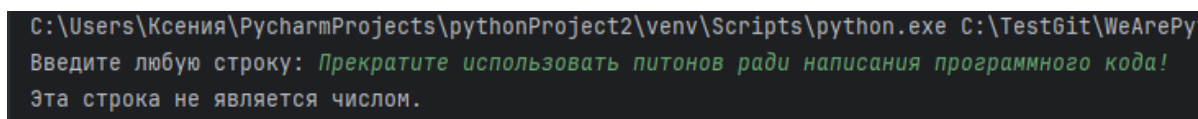
```
C:\Users\Ксения\PycharmProjects\pythonProject2\venv\Scripts\python.exe C:\TestGit\WeArePyth
Введите радиус цилиндра: 2
Введите высоту цилиндра: 4
Хотите получить только площадь не только боковой поверхности, но и цилиндра? (Да/Нет): Да
Площадь боковой поверхности цилиндра: 50.26548245743669
```

Рисунок 16.2 – Результат программы Task10.py



```
C:\Users\Ксения\PycharmProjects
Введите число: 4
Введите число: 5
Введите число: 2
Введите число: 8
Введите число: 0
320
```

Рисунок 16.3 – Результат программы Task12.py



```
C:\Users\Ксения\PycharmProjects\pythonProject2\venv\Scripts\python.exe C:\TestGit\WeArePyth
Введите любую строку: Прекратите использовать питонов ради написания программного кода!
Эта строка не является числом.
```

Рисунок 16.4 – Результат программы Task14.py

17. Приведите в отчете скриншоты работы программ решения индивидуального задания.

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

import sys

def get_person():
    """
    Запросить данные о личности.
    """
    # Запросить данные о личности.
    name = input("Фамилия и имя? ")
    zodiac_sign = input("Знак Зодиака? ")
    birth_date = input("Дата рождения? ")

    # Создать словарь.
    return {
        'name': name,
        'zodiac_sign': zodiac_sign,
        'birth_date': birth_date
    }

def display_persons(staff):
    """
    Отобразить список личностей.
    """
    # Проверить, что список личностей не пуст.
    if staff:
        # Заголовок таблицы.
        line = '+-{}-+-{}-+-{}-+-{}-+'.format(
            *args: '-' * 4,
            '-' * 30,
            '-' * 20,
            '-' * 8
        )
        print(line)
        print(
            '| {:^4} | {:^30} | {:^20} | {:^8} |'.format(
                *args: "№",
                "Имя",
            )
        )
```

Рисунок 17.1 – Код программы индивидуального задания INDIVIDual.py в IDE PyCharm

18. Зафиксировала сделанные изменения в репозитории.

```
C:\TestGit\WeArePythonistsAroudPeople>git add PyCharm
C:\TestGit\WeArePythonistsAroudPeople>git commit -m"add individual task"
[develop 1546ce2] add individual task
2 files changed, 144 insertions(+)
create mode 100644 PyCharm/Individual/INDIVIDual.py
```

Рисунок 18.1 – Коммит файлов в репозитории git

## Контрольные вопросы

1. Каково назначение функций в языке программирования Python?

Функции можно сравнить с небольшими программками, которые сами по себе, т. е. автономно, не исполняются, а встраиваются в обычную программу. Нередко их так и называют – подпрограммы. Других ключевых отличий функций от программ нет. Функции также при необходимости могут получать и возвращать данные. Только обычно они их получают не с ввода (клавиатуры, файла и др.), а из вызывающей программы. Сюда же они возвращают результат своей работы.

2. Каково назначение операторов `def` и `return`?

В языке программирования Python функции определяются с помощью оператора `def`. Выход из функции и передача данных в то место, откуда она была вызвана, выполняется оператором `return`.

3. Каково назначение локальных и глобальных переменных при написании функций в Python?

В программировании особое внимание уделяется концепции о локальных и глобальных переменных, а также связанное с ними представление об областях видимости. Соответственно, 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.

К глобальной переменной можно обратиться из локальной области видимости. К локальной переменной нельзя обратиться из глобальной области видимости, потому что локальная переменная существует только в момент выполнения тела функции. При выходе из нее, локальные переменные исчезают. Компьютерная память, которая под них отводилась, освобождается. Когда функция будет снова вызвана, локальные переменные будут созданы заново.

4. Как вернуть несколько значений из функции Python?

В Питоне позволительно возвращать из функции несколько объектов, перечислив их через запятую после команды `return`.

5. Какие существуют способы передачи значений в функцию?

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.

6. Как задать значение аргументов функции по умолчанию?

Для этого достаточно поставить знак равенства после имени параметра и указать его значение по умолчанию.

7. Каково назначение `lambda`-выражений в языке Python?

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.

8. Как осуществляется документирование кода согласно PEP257?

Документирование кода в python - достаточно важный аспект, ведь от нее порой зависит читаемость и быстрота понимания вашего кода, как другими людьми, так и вами через полгода. PEP 257 описывает соглашения, связанные со строками документации python, рассказывает о том, как нужно документировать python код. Цель этого PEP - стандартизировать структуру строк документации: что они должны в себя включать, и как это написать (не касаясь вопроса синтаксиса строк документации). Этот PEP описывает соглашения, а не правила или синтаксис.

9. В чем особенность однострочных и многострочных форм строк документации?

Многострочные документации состоят из сводной строки (`summary line`), имеющей такую же структуру, как и однострочный `docstring`, после которой следует пустая линия, а затем более сложное описание