

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»

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

(подпись)

Проверил: Воронкин Р. А.

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2023 г

Тема: Работа с функциями в языке Python

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

Ход работы:

1. Изучил теоретический материал работы.
2. Создал 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 * odik8 / **Repository name *** BSE11

✔ BSE11 is available.

Great repository names are short and memorable. Need inspiration? How about [fantastic-octo-succotash](#) ?

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

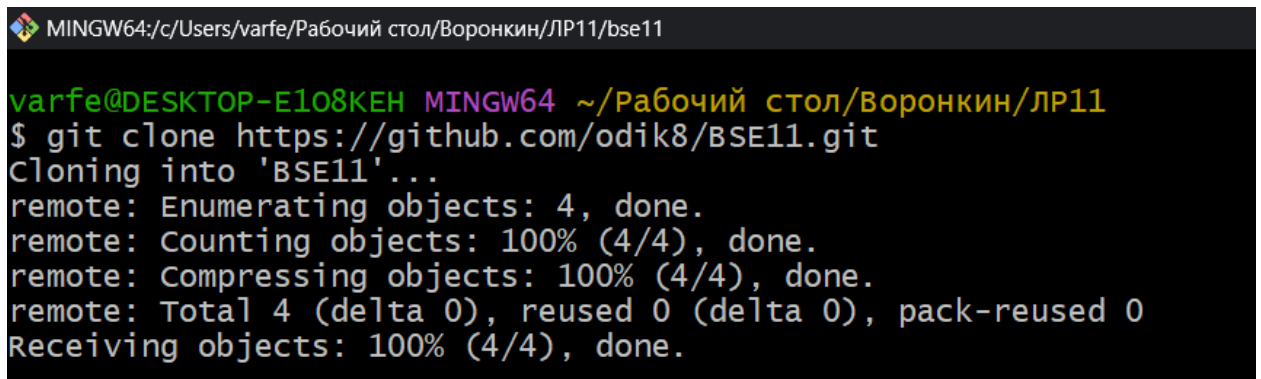
A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

You are creating a public repository in your personal account.

[Create repository](#)

Рисунок 1 – Создание репозитория

3. Выполнил клонирование созданного репозитория.

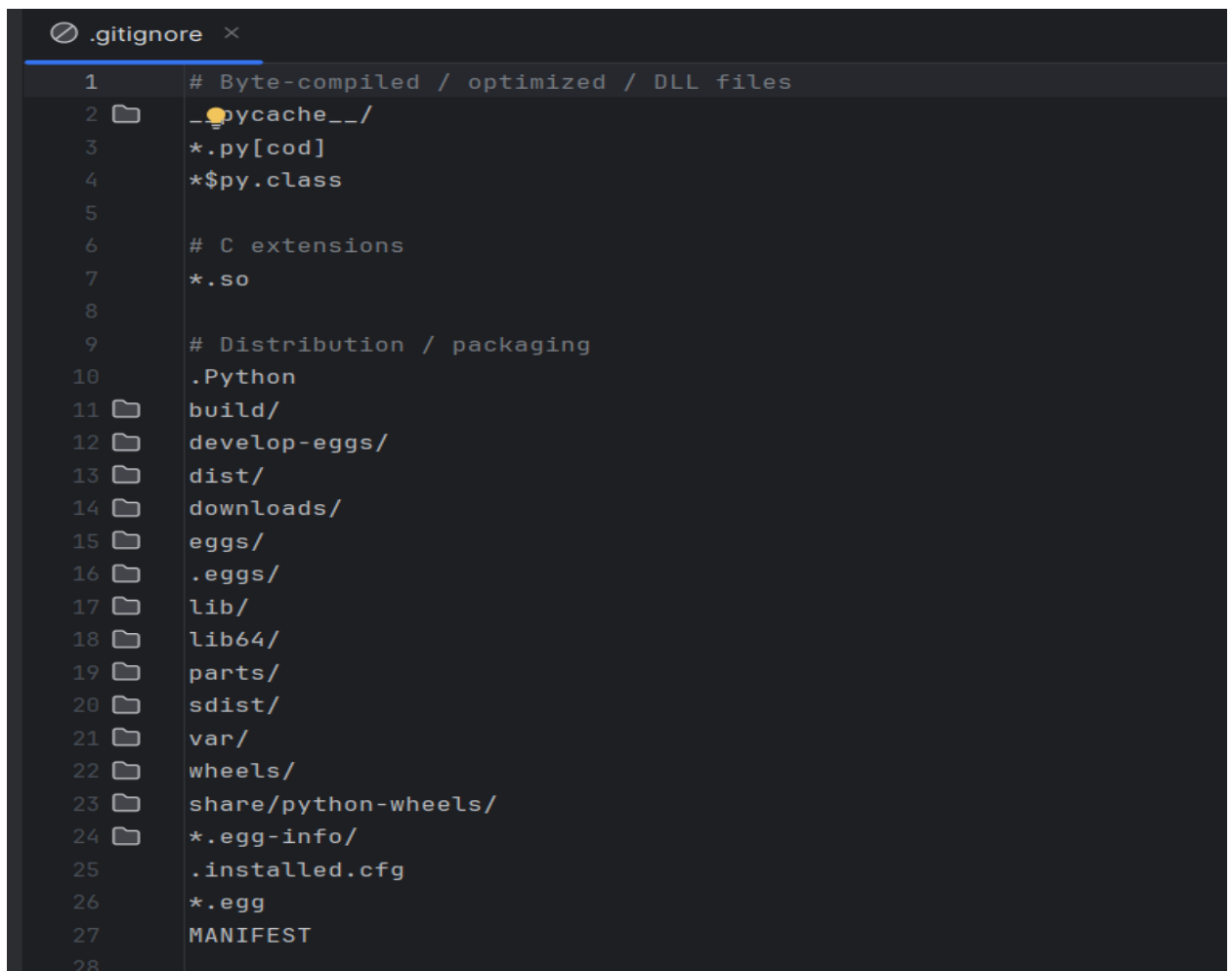


```
MINGW64:/c/Users/varfe/Рабочий стол/Воронкин/ЛР11/bse11

varfe@DESKTOP-E108KEN MINGW64 ~/Рабочий стол/Воронкин/ЛР11
$ git clone https://github.com/odik8/BSE11.git
Cloning into 'BSE11'...
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (4/4), done.
```

Рисунок 2 – Клонирование репозитория

4. Дополнил файл .gitignore необходимыми правилами для работы с IDE PyCharm.



```
.gitignore x
1 # Byte-compiled / optimized / DLL files
2 _pycache__/
3 *.py[cod]
4 *$py.class
5
6 # C extensions
7 *.so
8
9 # Distribution / packaging
10 .Python
11 build/
12 develop-eggs/
13 dist/
14 downloads/
15 eggs/
16 .eggs/
17 lib/
18 lib64/
19 parts/
20 sdist/
21 var/
22 wheels/
23 share/python-wheels/
24 *.egg-info/
25 .installed.cfg
26 *.egg
27 MANIFEST
28
```

Рисунок 3 – Файл .gitignore

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

```
varfe@DESKTOP-E108KEH MINGW64 ~/Рабочий стол/Воронкин/ЛР11/bse11 (main)
$ git flow init

Which branch should be used for bringing forth production releases?
  - main
Branch name for production releases: [main]
Branch name for "next release" development: [develop]

How to name your supporting branch prefixes?
Feature branches? [feature/]
Bugfix branches? [bugfix/]
Release branches? [release/]
Hotfix branches? [hotfix/]
Support branches? [support/]
Version tag prefix? []
Hooks and filters directory? [C:/Users/varfe/Рабочий стол/Воронкин/ЛР11/bse11/.git/hooks]
```

Рисунок 4 – Инициализация git-flow

6. Решил 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.py x
1  #!/usr/bin/env python3
2
3  1 usage new *
4  def positive():
5      print("Положительное")
6
7  1 usage new *
8  def negative():
9      print("Отрицательное")
10
11 1 usage new *
12 def test():
13     num = int(input("Введите целое число: "))
14     if num > 0:
15         positive()
16     elif num < 0:
17         negative()
18
19 if __name__ == "__main__":
20     test()
21
```

Рисунок 5 – Код решения первой задачи

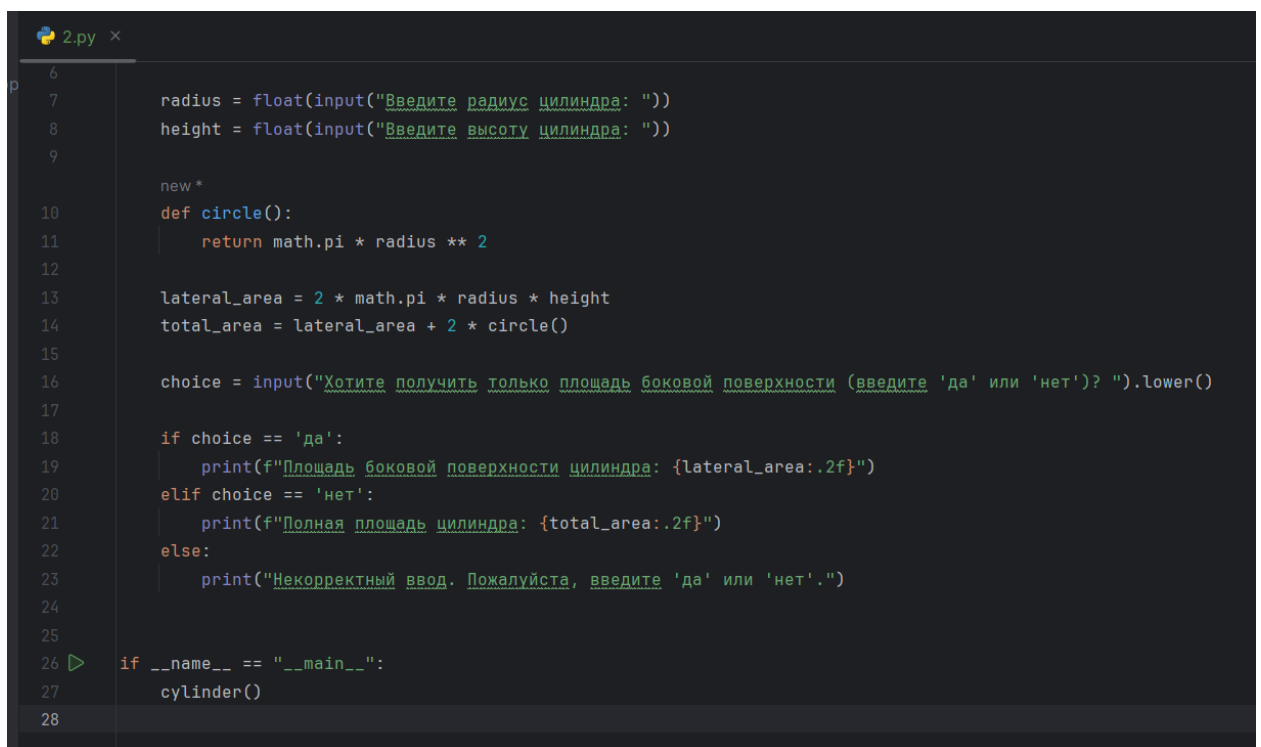
```
Run 1 x
C:\Users\varfe\AppData\Local\Programs\Python\Python39\python.exe
Введите целое число: -1
Отрицательное
Process finished with exit code 0
```

Рисунок 6 – Результат выполнения первой программы

В данном примере код будет работать корректно, даже если поменять местами определения функций **positive()** и **negative()**.

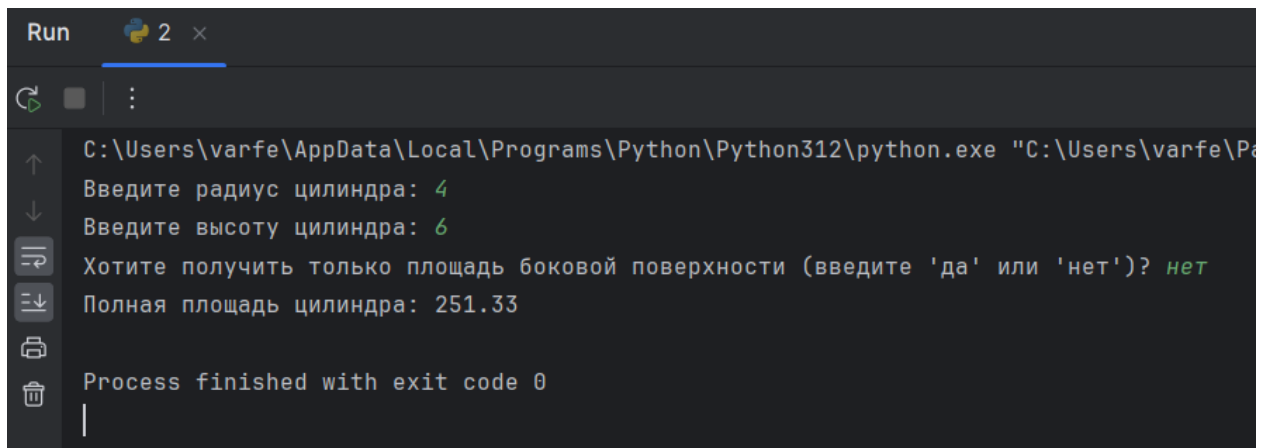
Таким образом, в Python порядок определения функций, если они вызываются после определения, не имеет значения. Однако, если вызов происходит до определения функции, то это может вызвать ошибку.

7. Решил 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()`.



```
2.py x
6
7     radius = float(input("Введите радиус цилиндра: "))
8     height = float(input("Введите высоту цилиндра: "))
9
10    new *
11    def circle():
12        return math.pi * radius ** 2
13
14    lateral_area = 2 * math.pi * radius * height
15    total_area = lateral_area + 2 * circle()
16
17    choice = input("Хотите получить только площадь боковой поверхности (введите 'да' или 'нет')? ").lower()
18
19    if choice == 'да':
20        print(f"Площадь боковой поверхности цилиндра: {lateral_area:.2f}")
21    elif choice == 'нет':
22        print(f"Полная площадь цилиндра: {total_area:.2f}")
23    else:
24        print("Некорректный ввод. Пожалуйста, введите 'да' или 'нет'.")
25
26  if __name__ == "__main__":
27      cylinder()
28
```

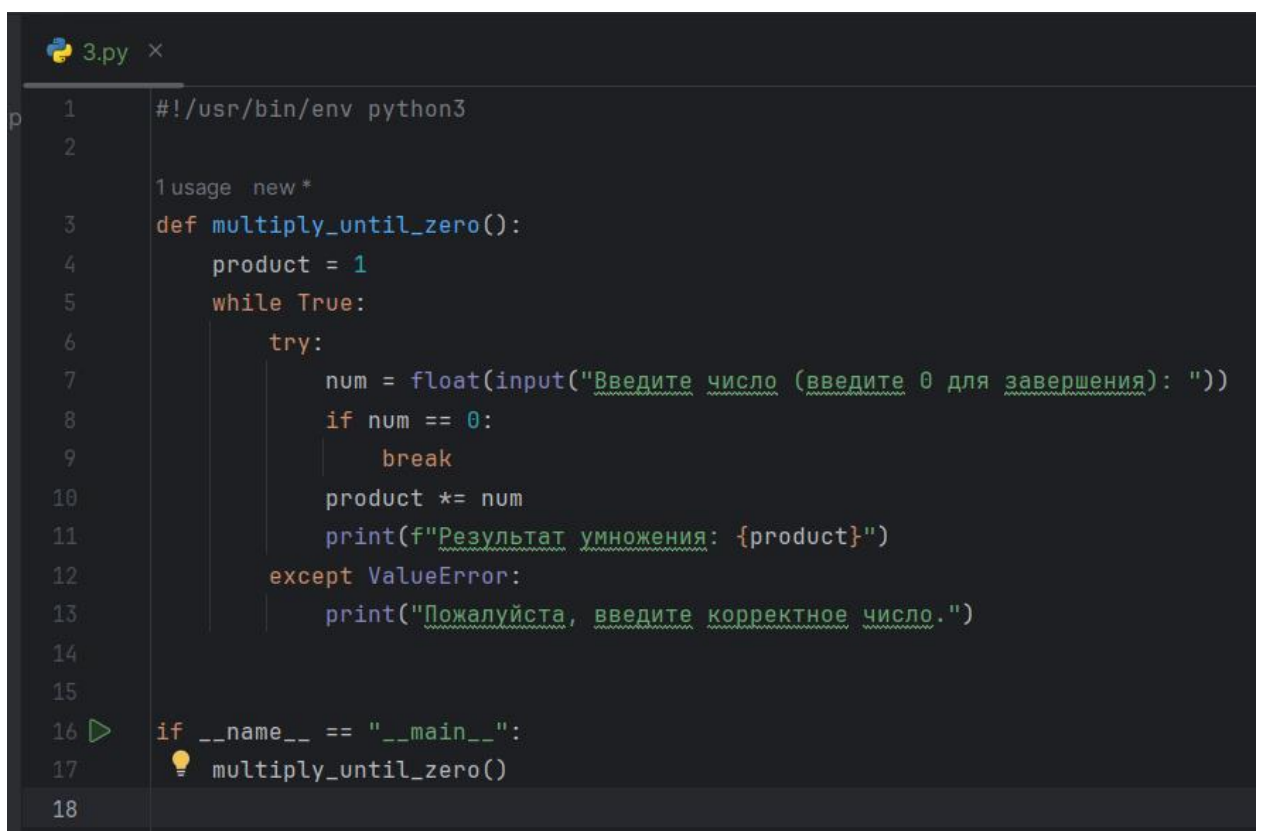
Рисунок 7 – Код решения второй задачи



```
Run 2 x
C:\Users\varfe\AppData\Local\Programs\Python\Python312\python.exe "C:\Users\varfe\Pa
Введите радиус цилиндра: 4
Введите высоту цилиндра: 6
Хотите получить только площадь боковой поверхности (введите 'да' или 'нет')? нет
Полная площадь цилиндра: 251.33
Process finished with exit code 0
```

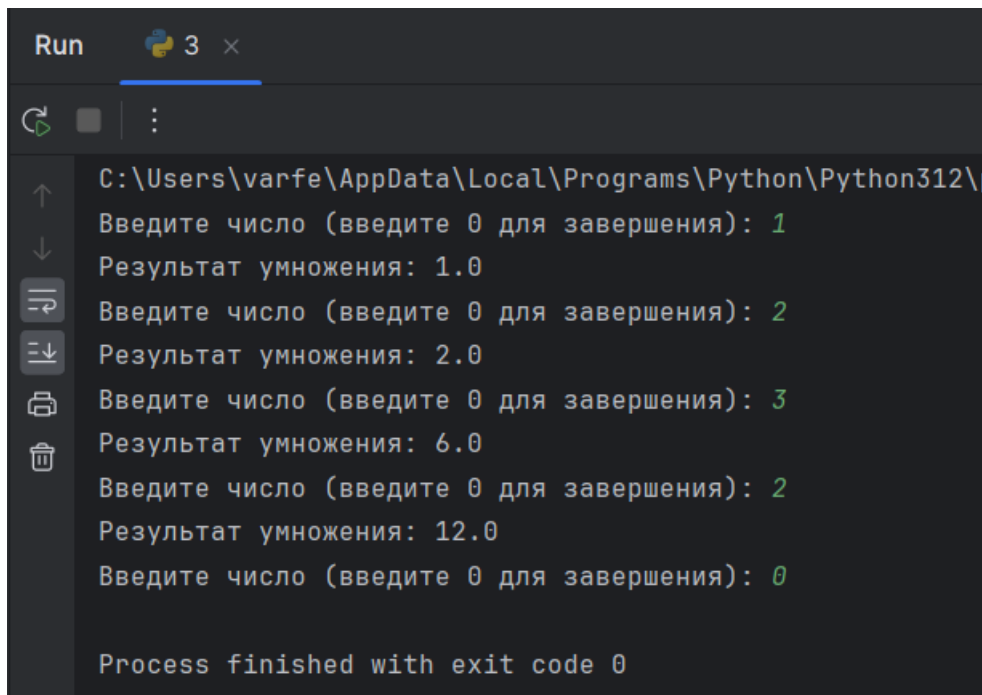
Рисунок 8 – Результат выполнения второй программы

8. 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.



```
3.py x
1  #!/usr/bin/env python3
2
3  usage new*
4  def multiply_until_zero():
5      product = 1
6      while True:
7          try:
8              num = float(input("Введите число (введите 0 для завершения): "))
9              if num == 0:
10                 break
11             product *= num
12             print(f"Результат умножения: {product}")
13         except ValueError:
14             print("Пожалуйста, введите корректное число.")
15
16 if __name__ == "__main__":
17     multiply_until_zero()
18
```

Рисунок 9 – Код решения третьей задачи



```
Run 3 x
C:\Users\varfe\AppData\Local\Programs\Python\Python312\
Введите число (введите 0 для завершения): 1
Результат умножения: 1.0
Введите число (введите 0 для завершения): 2
Результат умножения: 2.0
Введите число (введите 0 для завершения): 3
Результат умножения: 6.0
Введите число (введите 0 для завершения): 2
Результат умножения: 12.0
Введите число (введите 0 для завершения): 0
Process finished with exit code 0
```

Рисунок 10 – Результат выполнения третьей программы

9. Решил 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.


```
4.py x
1  #!/usr/bin/env python3
2
3  1 usage new *
4  def get_input():
5      return str(input("Введите строку: "))
6
7  1 usage new *
8  def test_input(value):
9      try:
10         int(value)
11         return True
12     except ValueError:
13         return False
14
15  1 usage new *
16  def str_to_int(value):
17      try:
18         result = int(value)
19         return result
20     except ValueError:
21         return None
22
23  1 usage new *
24  def print_int(value):
25      print(value)
26
27  if __name__ == "__main__":
28      input_value = get_input()
29
30      if test_input(input_value):
31         converted_value = str_to_int(input_value)
32
33         if converted_value is not None:
34             print_int(converted_value)
35         else:
36             print("Не удалось выполнить преобразование.")
37     else:
38         print("Введенное значение не может быть преобразовано в целое число.")
39
```

Рисунок 11 – Код решения четвертой задачи

```

PS C:\Users\varfe\Рабочий стол\Воронкин\ЛР11\BSE11> python3 4.py
Введите строку: 123
123
PS C:\Users\varfe\Рабочий стол\Воронкин\ЛР11\BSE11> python3 4.py
Введите строку: qwe
Введенное значение не может быть преобразовано в целое число.
PS C:\Users\varfe\Рабочий стол\Воронкин\ЛР11\BSE11>

```

Рисунок 12 – Результат выполнения четвертой программы

Индивидуальное задание: решить индивидуальное задание лабораторной работы 2.6, оформив каждую команду в виде отдельной функции.

Код решения:

```

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

from datetime import datetime

def get_birthdate():
    while True:
        try:
            date_str = input("Введите дату рождения в формате ДД.ММ.ГГГГ: ")
            birthdate = datetime.strptime(date_str, "%d.%m.%Y").date()
            return birthdate
        except ValueError:
            print("Ошибка Неправильный формат даты. Попробуйте снова.")

def add_person(list_of_people):
    print("\nДобавление нового контакта:")
    last_name = input("Введите фамилию: ")
    first_name = input("Введите имя: ")
    phone_number = input("Введите номер телефона: ")
    birthdate = get_birthdate()

    person = {
        'фамилия': last_name,
        'имя': first_name,
        'номер телефона': phone_number,
        'дата рождения': birthdate
    }

    list_of_people.append(person)
    list_of_people.sort(key=lambda x: x['дата рождения'])
    print("Человек добавлен\n")

def find_person_by_phone(list_of_people, phone):
    for person in list_of_people:
        if person['номер телефона'] == phone:
            return person

```

```

        return None

def print_person(list_of_people):
    if list_of_people:
        print("\nИнформация о человеке:")
        print(f"Фамилия: {list_of_people['фамилия']}")
        print(f"Имя: {list_of_people['имя']}")
        print(f"Номер телефона: {list_of_people['номер телефона']}")
        print(f"Дата рождения: {list_of_people['дата рождения'].strftime('%d.%m.%Y')}\n")
    else:
        print("Человек не найден.\n")

def main():
    list_of_people = []

    while True:
        print("1. Добавить человека")
        print("2. Найти человека по номеру телефона")
        print("3. Вывести список людей")
        print("4. Выйти")
        choice = input("Выберите действие (1/2/3/4): ")

        if choice == '1':
            add_person(list_of_people)
        elif choice == '2':
            phone_to_find = input("Введите номер телефона для поиска: ")
            found_person = find_person_by_phone(list_of_people,
phone_to_find)
            print_person(found_person)
        elif choice == '3':
            for _ in list_of_people: print_person(_)
        elif choice == '4':
            print("Программа завершена.")
            break
        else:
            print("Некорректный ввод. Попробуйте снова.")

if __name__ == "__main__":
    main()

```

Результат выполнения программы:

```

C:\Users\varfe\AppData\Local\Programs\Python\Python312\python.exe
"C:\Users\varfe\Рабочий стол\Воронкин\JP9\BSE9\individual.py"
1. Добавить человека
2. Найти человека по номеру телефона
3. Вывести список людей
4. Выйти
Выберите действие (1/2/3/4): 1

Добавление нового контакта:
Введите фамилию: Депп
Введите имя: Джонни
Введите номер телефона: 345
Введите дату рождения в формате ДД.ММ.ГГГГ: 09.06.1963
Контакт успешно добавлен

```

```
1. Добавить человека
2. Найти человека по номеру телефона
3. Вывести список людей
4. Выйти
Выберите действие (1/2/3/4): 1

Добавление нового контакта:
Введите фамилию: Вихорьков
Введите имя: Игорь
Введите номер телефона: 123
Введите дату рождения в формате ДД.ММ.ГГГГ: 29.12.1960
Контакт успешно добавлен

1. Добавить человека
2. Найти человека по номеру телефона
3. Вывести список людей
4. Выйти
Выберите действие (1/2/3/4): 2
Введите номер телефона для поиска: 345

Информация о человеке:
Фамилия: Депп
Имя: Джонни
Номер телефона: 345
Дата рождения: 09.06.1963

1. Добавить человека
2. Найти человека по номеру телефона
3. Вывести список людей
4. Выйти
Выберите действие (1/2/3/4): 3

Информация о человеке:
Фамилия: Вихорьков
Имя: Игорь
Номер телефона: 123
Дата рождения: 29.12.1960

Информация о человеке:
Фамилия: Депп
Имя: Джонни
Номер телефона: 345
Дата рождения: 09.06.1963

1. Добавить человека
2. Найти человека по номеру телефона
3. Вывести список людей
4. Выйти
Выберите действие (1/2/3/4):
```

Вопросы для защиты работы:

1. **Назначение функций в языке программирования Python:** Функции в Python предназначены для группировки повторяющегося кода, облегчения понимания и управления программой. Они могут принимать аргументы, выполнять определенные действия и возвращать результат.

2. Назначение операторов **def** и **return**:

- **def**: используется для определения функций. Создает объект-функцию, присваивая ему имя.
- **return**: используется для возврата значения из функции. Когда оператор **return** выполняется, функция завершает выполнение и возвращает указанное значение.

3. Назначение локальных и глобальных переменных при написании функций в Python:

- **Локальные переменные**: Ограничены областью видимости функции, в которой они определены. Существуют только внутри этой функции и не доступны извне.
- **Глобальные переменные**: объявляются за пределами функций и могут использоваться как внутри функций, так и вне их. Однако, для изменения глобальной переменной изнутри функции, нужно использовать ключевое слово **global**.

4. Как вернуть несколько значений из функции Python: Функция может возвращать кортеж, содержащий несколько значений. Пример:

```
def multiple_values(): return 1, 2, 3  
result = multiple_values()  
print(result) # Вывод:  
(1, 2, 3)
```

5. Какие существуют способы передачи значений в функцию:

- По позиции (позиционные аргументы).
- По имени (именованные аргументы).
- С использованием значений по умолчанию.
- С использованием переменного числа аргументов (***args**, ****kwargs**).

6. **Как задать значение аргументов функции по умолчанию:** можно указать значения по умолчанию в определении функции, например:

```
def example_function(arg1, arg2=10, arg3="default"): # Тело функции
```

7. **Назначение lambda-выражений в языке Python:** Lambda-выражения представляют собой анонимные (безымянные) функции, которые могут содержать только одно выражение. Они часто используются для создания простых функций в одной строке кода.

8. **Как осуществляется документирование кода согласно PEP257:** PEP257 определяет стандарты для документирования строк в Python-коде. Документирующие строки обычно располагаются в начале модуля, класса или функции и предоставляют информацию о их использовании. Для функций документирующая строка следует сразу после строки с определением функции.

9. **В чем особенность однострочных и многострочных форм строк документации:**

- **Однострочные строки документации (docstring):** обычно используются для краткого описания функции, класса или модуля. Размещаются внутри тройных одинарных или двойных кавычек.
- **Многострочные строки документации:** позволяют предоставить более подробное описание. Размещаются в начале функции, класса или модуля и могут включать детальные сведения, примеры использования и т.д.