

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 №2.8
дисциплины «Основы программной инженерии»

Выполнил:
Соколов Михаил Романович
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:
Богданов С.С., ассистент кафедры
инфокоммуникаций

(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 версии 3.x.

Ход выполнения работы:

1. Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT и .gitignore файл для языка 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 ().*

Repository template

No template ▾

Start your repository with a template repository's contents.

Owner * PoTtaTto ▾ / **Repository name *** fundamentals_of_software

✔ fundamentals_of_software_engineering_lab2_8 is available.

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

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.](#)

This will set `main` as the default branch. Change the default name in your [settings](#).

① You are creating a public repository in your personal account.

Create repository

Рисунок 1 – Создание репозитория с заданными настройками

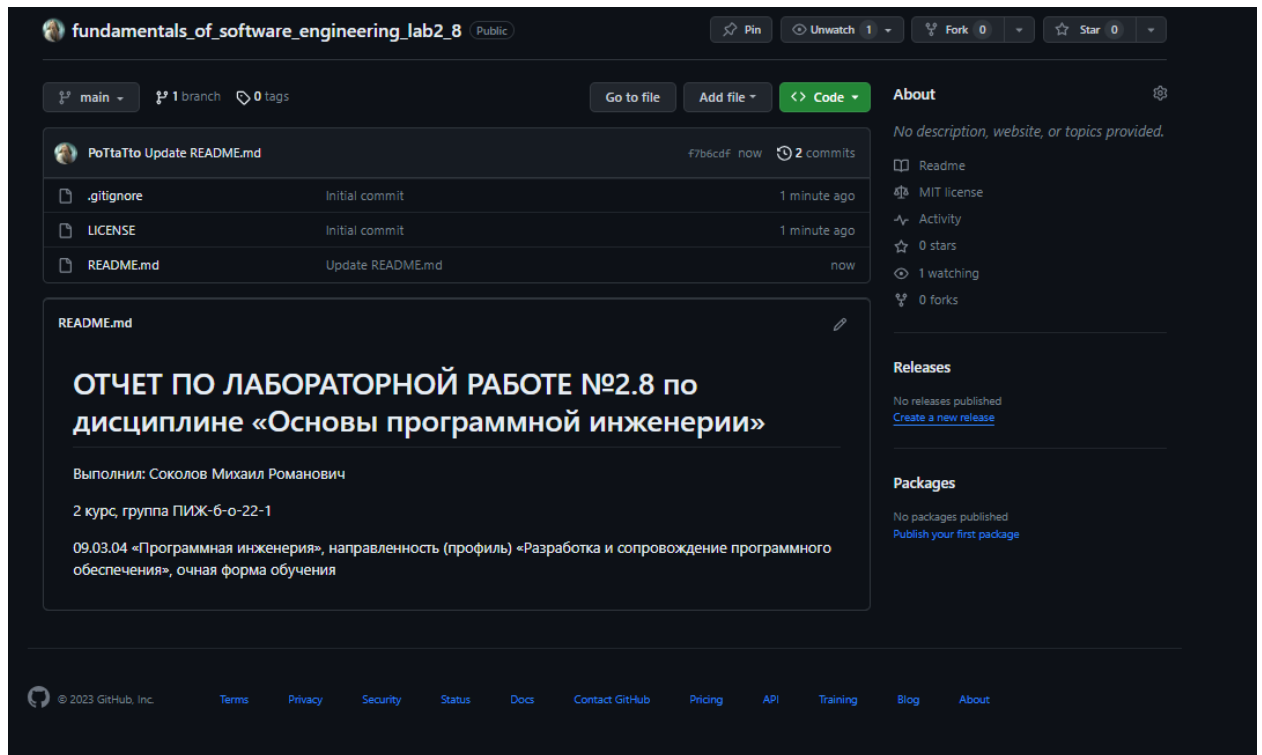


Рисунок 2 – Созданный репозиторий

```
PS C:\Study\СКФУ\Основы программной инженерии\Лабораторная работа 2.8> git clone https://github.com/PoTtaTto/fundamentals_of_software_engineering_lab2_8
Cloning into 'fundamentals_of_software_engineering_lab2_8'...
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
Receiving objects: 100% (8/8), done.
Resolving deltas: 100% (1/1), done.
PS C:\Study\СКФУ\Основы программной инженерии\Лабораторная работа 2.8> |
```

Рисунок 3 – Клонирование репозитория

```
PS C:\Study\СКФУ\Основы программной инженерии\Лабораторная работа 2.8> git checkout -b develop
Switched to a new branch 'develop'
PS C:\Study\СКФУ\Основы программной инженерии\Лабораторная работа 2.8> |
```

Рисунок 4 – Создание ветки develop, где будут происходить изменения проекта до его полного релиза

```
.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/
```

Рисунок 5 – Часть .gitignore файла, созданного GitHub

2. Проработаем пример лабораторной работы, фиксируя изменения.

Создадим для примера отдельный модуль:

Оформление каждой команды в виде вызова отдельной функции примера из лабораторной работы №2.6:

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

import sys
from datetime import date

def get_worker():
    """
    Запросить данные о работнике.
    """
    name = input("Фамилия и инициалы? ")
    post = input("Должность? ")
    year = int(input("Год поступления? "))
    # Создать словарь.
    return {
        'name': name,
        'post': post,
```

```

        'year': year,
    }

def display_workers(staff):
    """
    Отобразить список работников.
    """
    # Проверить, что список работников не пуст.
    if staff:
        # Заголовки таблицы.
        line = '+-{}-+-{}-+-{}-+-{}-+'.format(
            '-' * 4,
            '-' * 30,
            '-' * 20,
            '-' * 8
        )
        print(line)
        print(
            '| {:^4} | {:^30} | {:^20} | {:^8} |'.format(
                "№",
                "Ф.И.О.",
                "Должность",
                "Год"
            )
        )
        print(line)
        # Вывести данные о всех сотрудниках.
        for idx, worker in enumerate(staff, 1):
            print(
                '| {:>4} | {:<30} | {:<20} | {:>8} |'.format(
                    idx,
                    worker.get('name', ''),
                    worker.get('post', ''),
                    worker.get('year', 0)
                )
            )
            print(line)
    else:
        print("Список работников пуст.")

def select_workers(staff, period):
    """
    Выбрать работников с заданным стажем.
    """
    # Получить текущую дату.
    today = date.today()
    # Сформировать список работников.
    result = []
    for employee in staff:
        if today.year - employee.get('year', today.year) >= period:
            result.append(employee)
    # Возвратить список выбранных работников.
    return result

def main():
    """
    Главная функция программы.

```

```

"""
# Список работников.
workers = []
# Организовать бесконечный цикл запроса команд.
while True:
    # Запросить команду из терминала.
    command = input(">>> ").lower()
    # Выполнить действие в соответствии с командой.
    if command == 'exit':
        break
    elif command == 'add':
        # Запросить данные о работнике.
        worker = get_worker()
        # Добавить словарь в список.
        workers.append(worker)
        # Отсортировать список в случае необходимости.
        if len(workers) > 1:
            workers.sort(key=lambda item: item.get('name', ''))
    elif command == 'list':
        # Отобразить всех работников.
        display_workers(workers)
    elif command.startswith('select '):
        # Разбить команду на части для выделения стажа.
        parts = command.split(' ', maxsplit=1)
        # Получить требуемый стаж.
        period = int(parts[1])
        # Выбрать работников с заданным стажем.
        selected = select_workers(workers, period)
        # Отобразить выбранных работников.
        display_workers(selected)
    elif command == 'help':
        # Вывести справку о работе с программой.
        print("Список команд:\n")
        print("add - добавить работника;")
        print("list - вывести список работников;")
        print("select <стаж> - запросить работников со стажем;")
        print("help - отобразить справку;")
        print("exit - завершить работу с программой.")
    else:
        print(f"Неизвестная команда {command}", file=sys.stderr)

if __name__ == '__main__':
    main()

```

Листинг 1 – Код примера лабораторной работы

```
>>> help
Список команд:

add - добавить работника;
list - вывести список работников;
select <стаж> - запросить работников со стажем;
help - отобразить справку;
exit - завершить работу с программой.
>>> |
```

Рисунок 6 – Команда help

```
>>> exit

Process finished with exit code 0
|
```

Рисунок 7 – Команда exit

```
инженерии (лабораторная работа 2.0 (10
>>> add
Фамилия и инициалы? Соколов М.Р.
Должность? Студент
Год поступления? 2022
>>>
```

Рисунок 8 – Команда add

```
>>> list
```

№	Ф.И.О.	Должность	Год
1	Соколов М.Р.	Студент	2022

Рисунок 9 – Команда list

```
>>> select 1
```

№	Ф.И.О.	Должность	Год
1	Соколов М.Р.	Студент	2022

Рисунок 10 – Команда select (1)

```
>>> select 5
Список работников пуст.
```

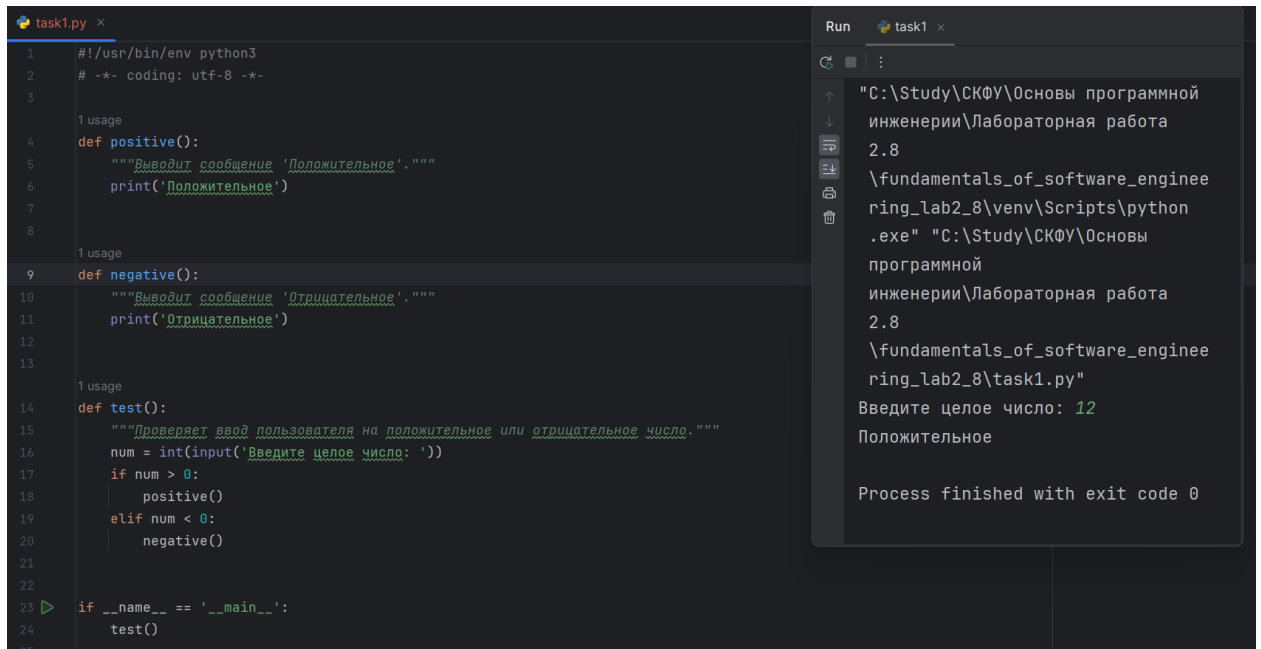
Рисунок 11 – Команда select (2)

```
>>> command
>>> Неизвестная команда command
```

Рисунок 12 – Ввод неизвестной команды

3. 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:



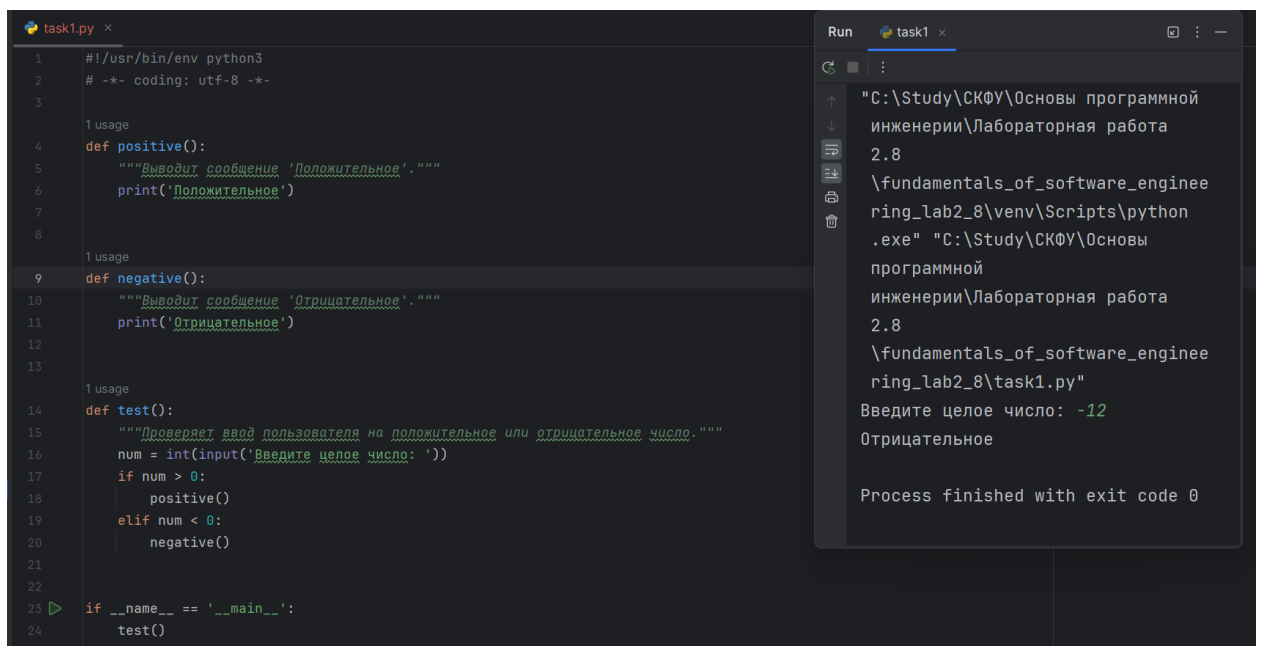
```
task1.py x
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  1 usage
5  def positive():
6      """Выводит сообщение 'Положительное'."""
7      print('Положительное')
8
9  1 usage
10 def negative():
11     """Выводит сообщение 'Отрицательное'."""
12     print('Отрицательное')
13
14  1 usage
15 def test():
16     """Проверяет ввод пользователя на положительное или отрицательное число."""
17     num = int(input('Введите целое число: '))
18     if num > 0:
19         positive()
20     elif num < 0:
21         negative()
22
23  if __name__ == '__main__':
24     test()
25
```

Run task1 x

"C:\Study\СКФУ\Основы программной инженерии\Лабораторная работа 2.8
\\fundamentals_of_software_engineering_lab2_8\venv\Scripts\python.exe" "C:\Study\СКФУ\Основы программной инженерии\Лабораторная работа 2.8
\\fundamentals_of_software_engineering_lab2_8\task1.py"
Введите целое число: 12
Положительное

Process finished with exit code 0

Рисунок 13 – Код и его выполнение (1)



```
task1.py x
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  1 usage
5  def positive():
6      """Выводит сообщение 'Положительное'."""
7      print('Положительное')
8
9  1 usage
10 def negative():
11     """Выводит сообщение 'Отрицательное'."""
12     print('Отрицательное')
13
14  1 usage
15 def test():
16     """Проверяет ввод пользователя на положительное или отрицательное число."""
17     num = int(input('Введите целое число: '))
18     if num > 0:
19         positive()
20     elif num < 0:
21         negative()
22
23  if __name__ == '__main__':
24     test()
25
```

Run task1 x

"C:\Study\СКФУ\Основы программной инженерии\Лабораторная работа 2.8
\\fundamentals_of_software_engineering_lab2_8\venv\Scripts\python.exe" "C:\Study\СКФУ\Основы программной инженерии\Лабораторная работа 2.8
\\fundamentals_of_software_engineering_lab2_8\task1.py"
Введите целое число: -12
Отрицательное

Process finished with exit code 0

Рисунок 14 – Код и его выполнение (2)

```
task1.py x
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4
5  1 usage
6  def test():
7      """Проверяет ввод пользователя на положительное или отрицательное число."""
8      num = int(input('Введите целое число: '))
9      if num > 0:
10         positive()
11     elif num < 0:
12         negative()
13
14  1 usage
15  def positive():
16      """Выводит сообщение 'Положительное'."""
17      print('Положительное')
18
19  1 usage
20  def negative():
21      """Выводит сообщение 'Отрицательное'."""
22      print('Отрицательное')
23
24  if __name__ == '__main__':
25      test()
```

Run task1 x

```
"C:\Study\СКФУ\Основы программной инженерии\Лабораторная работа 2.8\nfundamentals_of_software_engineering_lab2_8\venv\Scripts\python.exe" "C:\Study\СКФУ\Основы программной инженерии\Лабораторная работа 2.8\nfundamentals_of_software_engineering_lab2_8\task1.py"
Введите целое число: 12
Положительное

Process finished with exit code 0
```

Рисунок 15 – При изменении порядка функций программа продолжает работать

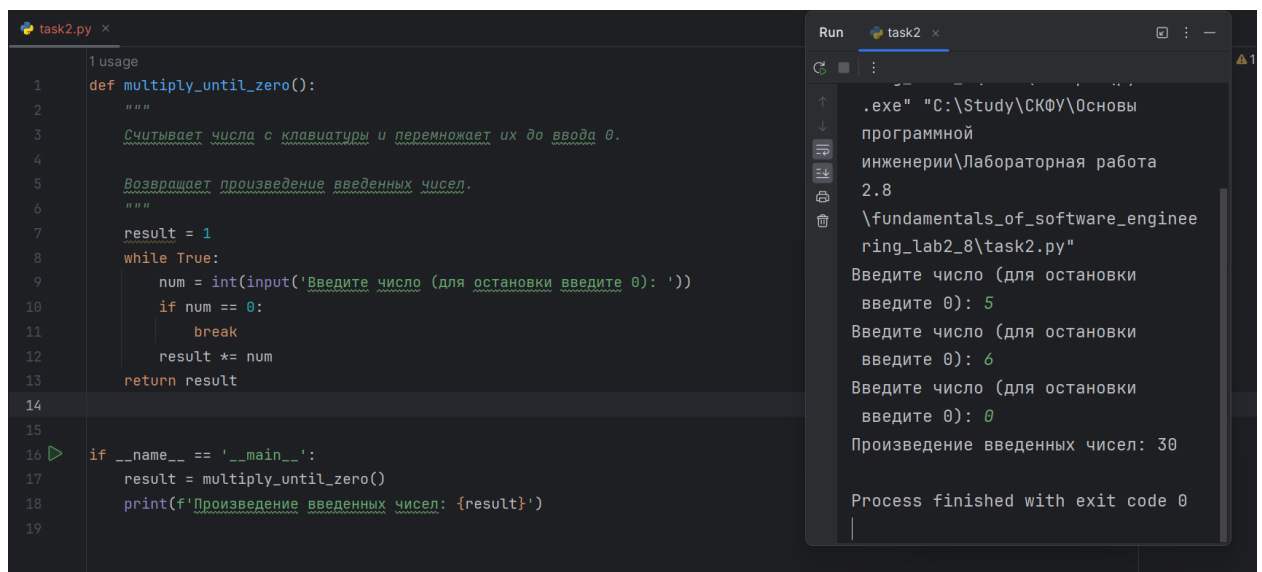
```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4
5  if __name__ == '__main__':
6      test()
7
8
9  def test():
10     """Проверяет ввод пользователя на положительное или отрицательное число."""
11     num = int(input('Введите целое число: '))
12     if num > 0:
13         positive()
14     elif num < 0:
15         negative()
16
17
18  1 usage
19  def positive():
20     """Выводит сообщение 'Положительное'."""
21     print('Положительное')
22
23  1 usage
24  def negative():
25     """Выводит сообщение 'Отрицательное'."""
26     print('Отрицательное')
```

Рисунок 16 – Однако попытка обратиться к функции test() до ее определения вызывает ошибку «Unresolved reference 'test'»

В первом случае, где `test()` вызывается после определения `positive()` и `negative()`, программа успешно выполняется. Это происходит потому, что на момент вызова `test()`, интерпретатор Python уже знает о существовании функций `positive()` и `negative()`.

Однако во втором случае, где `test()` вызывается до своего фактического определения, интерпретатор Python еще не знает о существовании `test()`, так как он еще не был определен к моменту выполнения. В результате возникает ошибка, поскольку вызов `test()` происходит до того, как функция фактически определена.

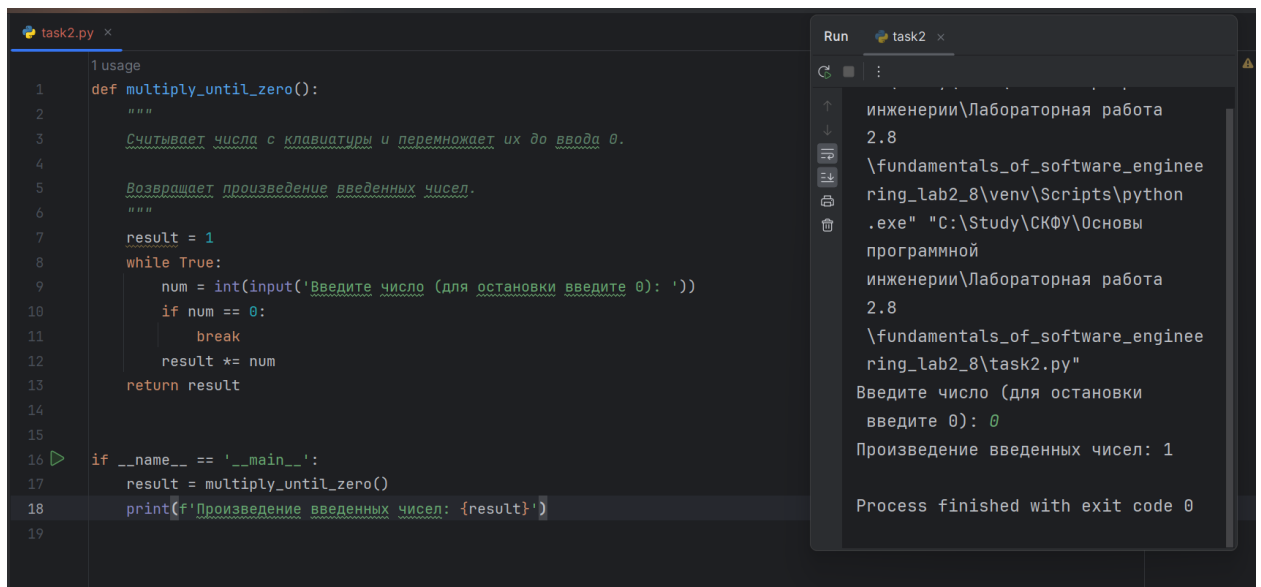
4. Решите следующую задачу: напишите функцию, которая считывает с клавиатуры числа и перемножает их до тех пор, пока не будет введен 0. Функция должна возвращать полученное произведение. Вызовите функцию и выведите на экран результат ее работы:



```
task2.py x
1 usage
2 def multiply_until_zero():
3     """
4     Считывает числа с клавиатуры и перемножает их до ввода 0.
5     Возвращает произведение введенных чисел.
6     """
7     result = 1
8     while True:
9         num = int(input('Введите число (для остановки введите 0): '))
10        if num == 0:
11            break
12        result *= num
13    return result
14
15
16 if __name__ == '__main__':
17     result = multiply_until_zero()
18     print(f'Произведение введенных чисел: {result}')
19
```

```
Run task2 x
.exe" "C:\Study\СКОУ\Основы
программной
инженерии\Лабораторная работа
2.8
\fundamentals_of_software_enginee
ring_lab2_8\task2.py"
Введите число (для остановки
введите 0): 5
Введите число (для остановки
введите 0): 6
Введите число (для остановки
введите 0): 0
Произведение введенных чисел: 30
Process finished with exit code 0
```

Рисунок 17 – Код и его выполнение (1)



```
task2.py x
1 usage
2 def multiply_until_zero():
3     """
4     Считывает числа с клавиатуры и перемножает их до ввода 0.
5     Возвращает произведение введенных чисел.
6     """
7     result = 1
8     while True:
9         num = int(input('Введите число (для остановки введите 0): '))
10        if num == 0:
11            break
12        result *= num
13    return result
14
15
16 if __name__ == '__main__':
17     result = multiply_until_zero()
18     print(f'Произведение введенных чисел: {result}')
19
```

```
Run task2 x
инженерии\Лабораторная работа
2.8
\\fundamentals_of_software_enginee
ring_lab2_8\venv\Scripts\python
.exe" "C:\Study\СКФУ\Основы
программной
инженерии\Лабораторная работа
2.8
\\fundamentals_of_software_enginee
ring_lab2_8\task2.py"
Введите число (для остановки
введите 0): 0
Произведение введенных чисел: 1
Process finished with exit code 0
```

Рисунок 18 – Код и его выполнение (2)

5. 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 . В теле `cylinder()` у пользователя спрашивается, хочет ли он получить только площадь боковой поверхности цилиндра, которая вычисляется по формуле , 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 circle(radius):
    """
    Вычисляет площадь круга.

    Args:
    - radius (float): Радиус круга.

    Returns:
    - float: Площадь круга.
    """
    return math.pi * radius ** 2

def cylinder():
```

```

"""
Вычисляет площадь боковой поверхности или полной площади цилиндра.

Returns:
- float: Площадь боковой поверхности цилиндра или полная площадь
цилиндра.
"""
radius = float(input('Введите радиус цилиндра: '))
height = float(input('Введите высоту цилиндра: '))

side_area = 2 * math.pi * radius * height
full_area = side_area + 2 * circle(radius)

choice = input('Хотите получить только боковую площадь? (да/нет):
').lower()
if choice == 'да':
    return side_area
elif choice == 'нет':
    return full_area
else:
    return 'Неправильный выбор.'

if __name__ == '__main__':
    result = cylinder()
    print(f'Площадь цилиндра: {result}')

```

Листинг 2 – Код задачи

```

"C:\Study\СКФУ\Основы программной инженерии\Лабораторная работа
2.8\fundamentals_of_software_engineering_lab2_8\venv\Scripts
\python.exe" "C:\Study\СКФУ\Основы программной
инженерии\Лабораторная работа 2
.8\fundamentals_of_software_engineering_lab2_8\task3.py"
Введите радиус цилиндра: 12
Введите высоту цилиндра: 5
Хотите получить только боковую площадь? (да/нет): нет
Площадь цилиндра: 1281.7698026646356

Process finished with exit code 0

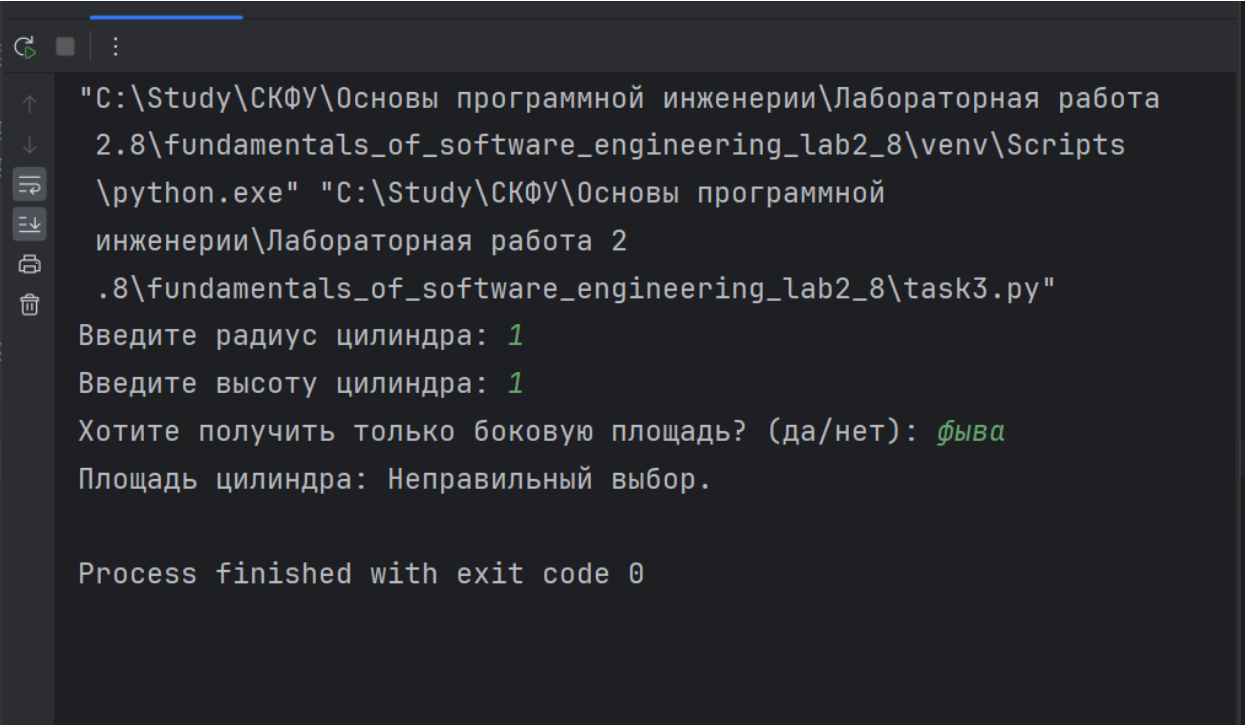
```

Рисунок 19 – Пример выполнения кода (1)

```
"C:\Study\СКФУ\Основы программной инженерии\Лабораторная работа
2.8\fundamentals_of_software_engineering_lab2_8\venv\Scripts
\python.exe" "C:\Study\СКФУ\Основы программной
инженерии\Лабораторная работа 2
.8\fundamentals_of_software_engineering_lab2_8\task3.py"
Введите радиус цилиндра: 5
Введите высоту цилиндра: 2
Хотите получить только боковую площадь? (да/нет): да
Площадь цилиндра: 62.83185307179586

Process finished with exit code 0
```

Рисунок 20 – Пример выполнения кода (2)



```
"C:\Study\СКФУ\Основы программной инженерии\Лабораторная работа
2.8\fundamentals_of_software_engineering_lab2_8\venv\Scripts
\python.exe" "C:\Study\СКФУ\Основы программной
инженерии\Лабораторная работа 2
.8\fundamentals_of_software_engineering_lab2_8\task3.py"
Введите радиус цилиндра: 1
Введите высоту цилиндра: 1
Хотите получить только боковую площадь? (да/нет): фыва
Площадь цилиндра: Неправильный выбор.

Process finished with exit code 0
```

Рисунок 21 – Пример выполнения кода (3)

6. 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:

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

```
def get_input():
    """
    Запрашивает ввод с клавиатуры и возвращает полученную строку.

    Returns:
    - str: Введенная строка.
    """
    user_input = input("Введите значение: ")
    return user_input

def test_input(value):
    """
    Проверяет, можно ли значение преобразовать к целому числу.

    Args:
    - value (str): Значение для проверки.

    Returns:
    - bool: True, если значение можно преобразовать в int, иначе False.
    """
    try:
        int(value)
        return True
    except ValueError:
        return False

def str_to_int(value):
    """
    Преобразует переданное значение к целочисленному типу.

    Args:
    - value (str): Значение для преобразования.

    Returns:
```

```

- int: Преобразованное целочисленное значение.
"""
return int(value)

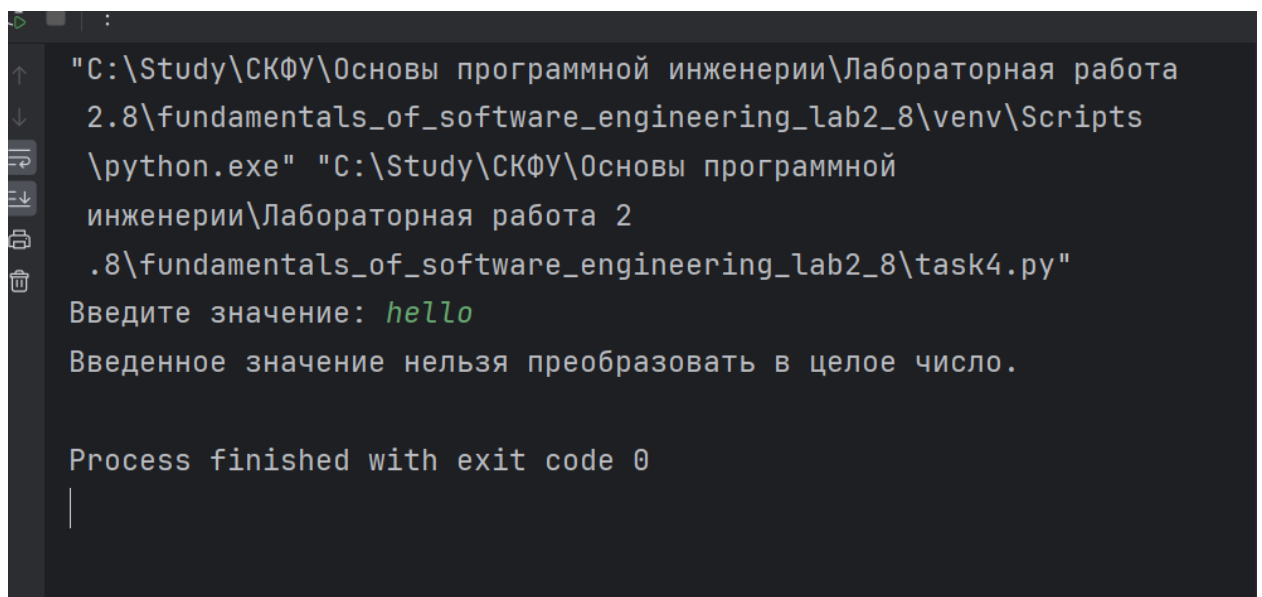
def print_int(number):
    """
    Выводит переданное значение на экран.

    Args:
    - number (int): Значение для вывода.
    """
    print(number)

if __name__ == '__main__':
    user_value = get_input()
    if test_input(user_value):
        int_value = str_to_int(user_value)
        print_int(int_value)
    else:
        print("Введенное значение нельзя преобразовать в целое число.")

```

Листинг 3 – Код задачи



```

"C:\Study\СКФУ\Основы программной инженерии\Лабораторная работа
2.8\fundamentals_of_software_engineering_lab2_8\venv\Scripts
\python.exe" "C:\Study\СКФУ\Основы программной
инженерии\Лабораторная работа 2
.8\fundamentals_of_software_engineering_lab2_8\task4.py"
Введите значение: hello
Введенное значение нельзя преобразовать в целое число.

Process finished with exit code 0

```

Рисунок 22 – Пример выполнения кода (1)


```
"C:\Study\СКФУ\Основы программной инженерии\Лабораторная работа
2.8\fundamentals_of_software_engineering_lab2_8\venv\Scripts
\python.exe" "C:\Study\СКФУ\Основы программной
инженерии\Лабораторная работа 2
.8\fundamentals_of_software_engineering_lab2_8\task4.py"
Введите значение: 1923
1923

Process finished with exit code 0
```

Рисунок 23 – Пример выполнения кода (2)

```
"C:\Study\СКФУ\Основы программной инженерии\Лабораторная работа
2.8\fundamentals_of_software_engineering_lab2_8\venv\Scripts
\python.exe" "C:\Study\СКФУ\Основы программной
инженерии\Лабораторная работа 2
.8\fundamentals_of_software_engineering_lab2_8\task4.py"
Введите значение: -100500
-100500

Process finished with exit code 0
```

Рисунок 24 – Пример выполнения кода (3)

7. Решение индивидуального задания (вариант №7): решить индивидуальное задание лабораторной работы 2.6, оформив каждую команду в виде отдельной функции:

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

import sys
from random import randint
```

```

def add_train(trains):
    """
    Добавляет информацию о поезде в список trains.

    Args:
    - trains (list): Список поездов.

    """
    train_num = int(input('Введите номер поезда: '))
    destination = input('Введите пункт назначения: ')
    start_time = input('Введите время выезда: ')
    trains.append({'num': train_num, 'destination': destination,
'start_time': start_time})
    if len(trains) > 1:
        trains.sort(key=lambda item: item['start_time'])

def list_trains(trains):
    """
    Выводит список поездов на экран.

    Args:
    - trains (list): Список поездов.

    """
    line = f'+-{"-" * 15}+-{"-" * 30}+-{"-" * 25}+'
    print(line)
    header = f"| {'№ поезда':^15} | {'Пункт назначения':^30} | {'Время
отъезда':^25} |"
    print(header)
    print(line)
    for train in trains:
        num = train.get('num', randint(1000, 10000))
        destination = train.get('destination', 'None')
        start_time = train.get('start_time', 'None')
        recording = f"| {num:^15} | {destination:^30} | {start_time:^25}
|"
        print(recording)
    print(line)

def select_train(trains, cmd_parts):
    """
    Выводит информацию о поездах, направляющихся в указанный пункт.

    Args:
    - trains (list): Список поездов.
    - cmd_parts (list): Список команды и параметра.

    """
    cmd_destination = cmd_parts[1]
    select_trains = [train for train in trains if train['destination'] ==
cmd_destination]
    if len(select_trains) > 1:
        for train in select_trains:
            print(f'{train["num"]:^15}: {train["start_time"]:^25}')
    else:
        print('Нет поездов едущих в данное место!', file=sys.stderr)

```

```

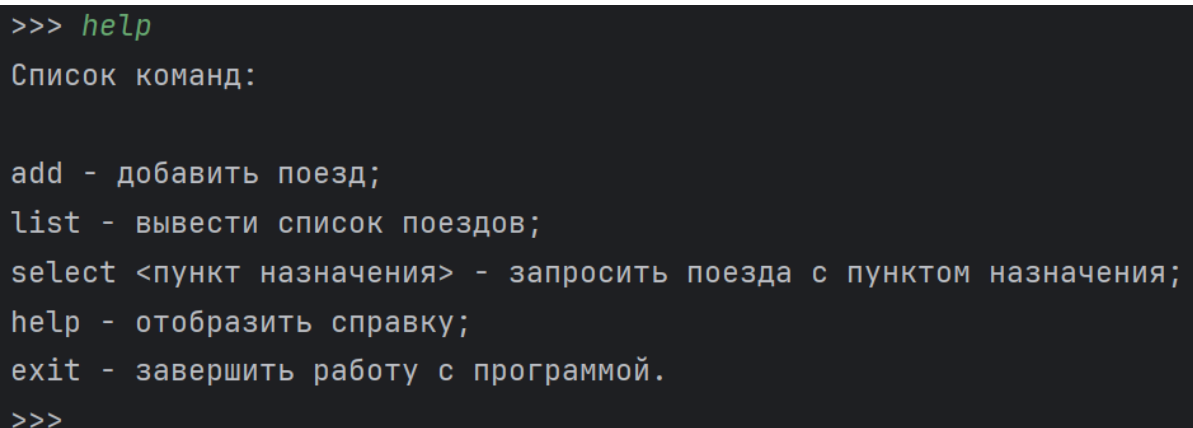
def show_help():
    """
    Выводит список доступных команд на экран.

    """
    print("Список команд:\n")
    print("add - добавить поезд;")
    print("list - вывести список поездов;")
    print("select <пункт назначения> - запросить поезда с пунктом
назначения;")
    print("help - отобразить справку;")
    print("exit - завершить работу с программой.")

if __name__ == '__main__':
    trains = []
    while True:
        cmd = input('>>> ')
        cmd_parts = cmd.split(maxsplit=1)
        match cmd_parts[0]:
            case 'add':
                add_train(trains)
            case 'list':
                list_trains(trains)
            case 'select':
                select_train(trains, cmd_parts)
            case 'help':
                show_help()
            case 'exit':
                break
            case _:
                print(f'Неизвестная команда {cmd}', file=sys.stderr)

```

Листинг 4 – Код задания



```

>>> help
Список команд:

add - добавить поезд;
list - вывести список поездов;
select <пункт назначения> - запросить поезда с пунктом назначения;
help - отобразить справку;
exit - завершить работу с программой.
>>>

```

Рисунок 25 – Команда help

```
>>> exit
```

```
Process finished with exit code 0
```

Рисунок 26 – Команда exit

```
>>> add
```

```
Введите номер поезда: 12
```

```
Введите пункт назначения: Ставрополь
```

```
Введите время выезда: 2023-11-26 14:00
```

```
>>> |
```

Рисунок 27 – Команда add

```
>>> list
```

+-----+-----+-----+		
№ поезда	Пункт назначения	Время отъезда
+-----+-----+-----+		
12	Ставрополь	2023-11-26 14:00
+-----+-----+-----+		

```
>>> |
```

Рисунок 28 – Команда list

```
>>> select Ставрополь
```

```
12 : 2023-11-26 14:00
```

```
>>> |
```

Рисунок 29 – Команда select (1)

```
>>> select Москва
>>> Нет поездов едущих в данное место!
```

Рисунок 30 – Команда select (2)

```
command
Неизвестная команда command
>>>
```

Рисунок 31 – Использование неизвестной команды

8. Сделаем merge веток main/master и отправим изменения на удаленный репозиторий:

```
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
PS C:\Study\СКФУ\Основы программной инженерии\Лабораторная работа 2.8\fundamentals_of_software_engineering_lab2_8> git merge develop
Updating f7b6cdf..d60ab95
Fast-forward
 .gitignore           | 2 +-
 individual_task1.py  | 95 +++++
 lab_task1.py         | 122 +++++
 task1.py             | 25 +++++
 task2.py             | 21 +++++
 task3.py             | 44 +++++
 task4.py             | 61 +++++
 7 files changed, 369 insertions(+), 1 deletion(-)
 create mode 100644 individual_task1.py
 create mode 100644 lab_task1.py
 create mode 100644 task1.py
 create mode 100644 task2.py
 create mode 100644 task3.py
 create mode 100644 task4.py
PS C:\Study\СКФУ\Основы программной инженерии\Лабораторная работа 2.8\fundamentals_of_software_engineering_lab2_8>
```

Рисунок 32 – merge веток main/develop

```
PS C:\Study\СКФУ\Основы программной инженерии\Лабораторная работа 2.8\fundamentals_of_software_engineering_lab2_8> git push origin main
Enumerating objects: 22, done.
Counting objects: 100% (22/22), done.
Delta compression using up to 12 threads
Compressing objects: 100% (20/20), done.
Writing objects: 100% (20/20), 5.79 KiB | 5.79 MiB/s, done.
Total 20 (delta 7), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (7/7), completed with 1 local object.
To https://github.com/PoItaTo/fundamentals_of_software_engineering_lab2_8
 f7b6cdf..d60ab95 main -> main
PS C:\Study\СКФУ\Основы программной инженерии\Лабораторная работа 2.8\fundamentals_of_software_engineering_lab2_8>
```

Рисунок 33 – Отправка изменений на удаленный репозиторий

Ответы на контрольные вопросы:

1. Назначение функций в Python:

Функции в Python используются для группировки кода в логические блоки, обеспечивая повторное использование, упрощение понимания кода и уменьшение дублирования. Они выполняют конкретные задачи и могут вызываться из других частей программы.

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

– `def`: используется для определения функции в Python. Он указывает на начало блока функции и ее имени.

– `return`: возвращает значение из функции и завершает ее выполнение. Оно передает результат обратно в вызывающую часть программы.

3. Назначение локальных и глобальных переменных:

– Локальные переменные: существуют только в пределах функции, в которой они определены. Их область видимости ограничена функцией, и они уничтожаются после завершения выполнения функции.

– Глобальные переменные: Определены вне функций и доступны во всей программе. Их можно использовать в любой части программы.

4. Возвращение нескольких значений из функции в Python:

Python позволяет вернуть несколько значений из функции, объединяя их в кортеж, списке или другой структуре данных и возвращая эту структуру с помощью оператора `return`.

5. Способы передачи значений в функцию:

– По значению (по умолчанию для неизменяемых типов данных): Передается копия значения.

– По ссылке (для изменяемых типов данных): передается ссылка на объект, позволяя изменять его внутри функции.

6. Задание значения аргументов функции по умолчанию:

В Python можно установить значения аргументов по умолчанию, указав их в определении функции. Это позволяет вызывать функцию без указания всех аргументов.

7. Назначение lambda-выражений:

Lambda-выражения используются для создания анонимных функций в Python. Они представляют собой короткий синтаксис для определения функций без использования ключевого слова `def`.

8. Документирование кода согласно PEP257:

Для документирования кода в Python используют строки документации (docstrings). Для описания функций, модулей или классов рекомендуется использовать строки документации в определении этих элементов.

9. Особенности однострочных и многострочных строк документации:

– Однострочные строки документации: представляют собой однострочный комментарий в тройных кавычках (`"` или `"""`). Используются для краткого описания функции, метода или модуля.

– Многострочные строки документации: занимают несколько строк и также записываются в тройных кавычках. Используются для подробного описания функций, методов, классов или модулей.