

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 «Программирование на Python»
Вариант 29

Выполнил:
Саенко Андрей Максимович
2 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и
вычислительная техника»,
направленность (профиль)
«Программное обеспечение средств
вычислительной техники и
автоматизированных систем», очная
форма обучения

(подпись)

Руководитель практики:
Воронкин Р.А., канд. технических
наук, доцент кафедры
инфокоммуникаций

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____
Ставрополь, 2023 г.

Тема: Работа с функциями в языке Python

Цель: приобретение навыков по работе с функциями при написании программ с помощью языка программирования Python версии 3.x

Порядок выполнения работы:

1. Проработаны примеры лабораторной работы

Пример 1. Для примера 1 лабораторной работы 2.6, оформить каждую команду в виде вызова отдельной функции.

Результат работы программы:

```
>>> add
Фамилия и инициалы? Иванов И.И.
Должность? Программист
Год поступления? 2001
>>> list
+-----+-----+-----+-----+
| № |          Ф.И.О.          |      Должность      |      Год      |
+-----+-----+-----+-----+
|  1 | Иванов И.И.              | Программист         |      2001     |
+-----+-----+-----+-----+
>>> 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 1 – Результат работы программы

Код программы:

```
#!/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:
        # Заголовок 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 соответствие 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)

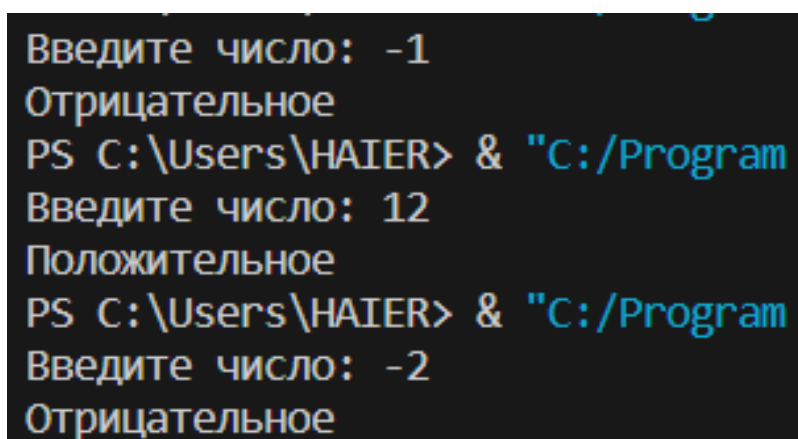
if __name__ == '__main__':
    main()
```

2. Решены задачи

1) 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.

Результат работы программы:



```
Введите число: -1
Отрицательное
PS C:\Users\NAIER> & "C:/Program
Введите число: 12
Положительное
PS C:\Users\NAIER> & "C:/Program
Введите число: -2
Отрицательное
```

Рисунок 2 – Результат работы программы

Код программы:

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

def positive():
    print("Положительное")

def negative():
    print("Отрицательное")

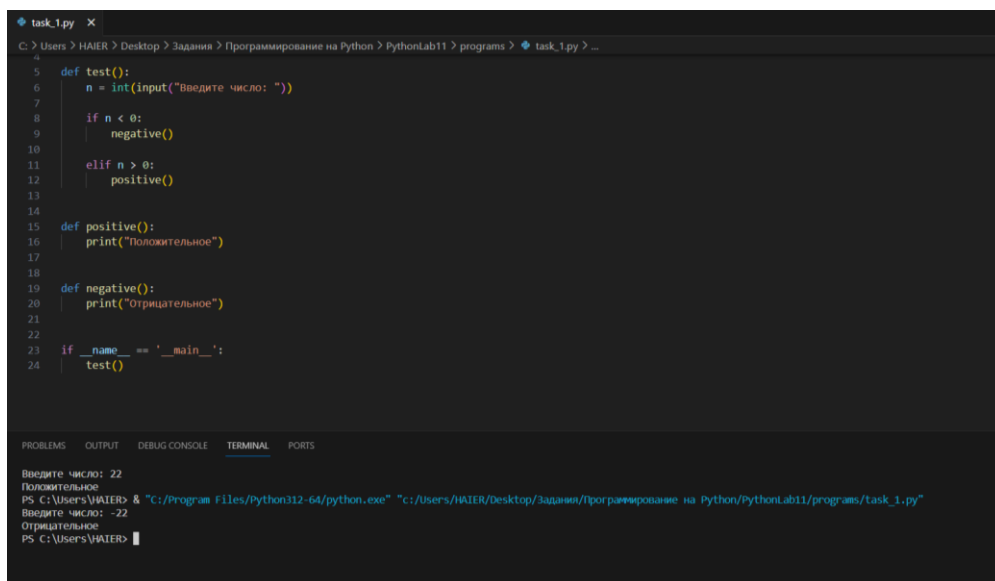
def test():
    n = int(input("Введите число: "))

    if n < 0:
        negative()

    elif n > 0:
        positive()

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

Программа продолжает корректно работать даже после того, как функции `negative()` и `positive()` были определены позже функции `test()`. Связано это с тем, что вызов функции `test()`, содержащей вызовы этих функций, происходит после их определения.



```
task_1.py
C:\Users\HAIER\Desktop\Задания\Программирование на Python\PythonLab11\programs> task_1.py > ...

5 def test():
6     n = int(input("Введите число: "))
7
8     if n < 0:
9         negative()
10
11    elif n > 0:
12        positive()
13
14
15 def positive():
16     print("Положительное")
17
18
19 def negative():
20     print("Отрицательное")
21
22
23 if __name__ == '__main__':
24     test()

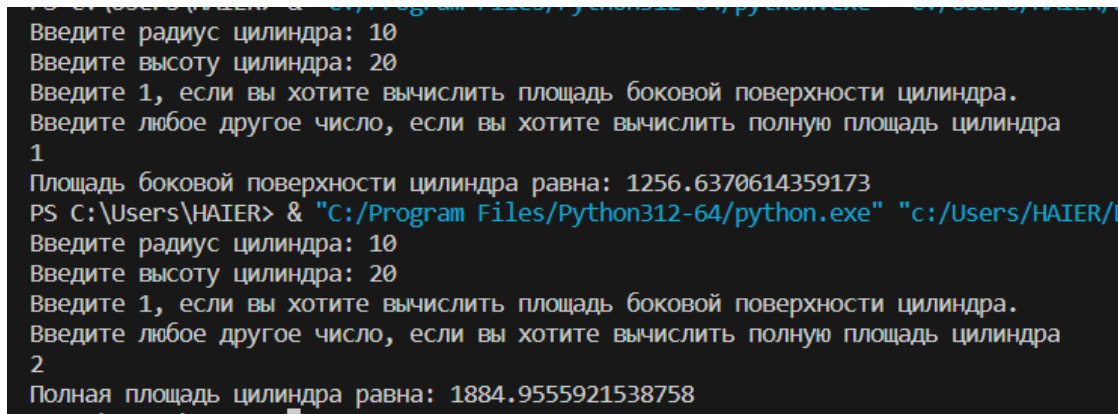
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Введите число: 22
Положительное
PS C:\Users\HAIER> & "C:/Program Files/Python312-64/python.exe" "C:/Users/HAIER/Desktop/Задания/Программирование на Python/PythonLab11/programs/task_1.py"
Введите число: -22
Отрицательное
PS C:\Users\HAIER>
```

Рисунок 3 – Результат работы программы

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 $S = \pi r^2$. 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()`.

Результат работы программы:



```
Введите радиус цилиндра: 10
Введите высоту цилиндра: 20
Введите 1, если вы хотите вычислить площадь боковой поверхности цилиндра.
Введите любое другое число, если вы хотите вычислить полную площадь цилиндра
1
Площадь боковой поверхности цилиндра равна: 1256.6370614359173
PS C:\Users\HAIER> & "C:/Program Files/Python312-64/python.exe" "c:/Users/HAIER/...
Введите радиус цилиндра: 10
Введите высоту цилиндра: 20
Введите 1, если вы хотите вычислить площадь боковой поверхности цилиндра.
Введите любое другое число, если вы хотите вычислить полную площадь цилиндра
2
Полная площадь цилиндра равна: 1884.9555921538758
```

Рисунок 4 – Результат работы программы

Код программы:

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

from math import pi

def circle(r):
    return pi*(r**2)

def cylinder():
    r = int(input("Введите радиус цилиндра: "))
    h = int(input("Введите высоту цилиндра: "))
    v = int(input("Введите 1, если вы хотите вычислить"+
        "площадь боковой поверхности цилиндра.\n"+
        "Введите любое другое число, если вы хотите"+
        "вычислить полную площадь цилиндра\n"))

    if v == 1:
```

```

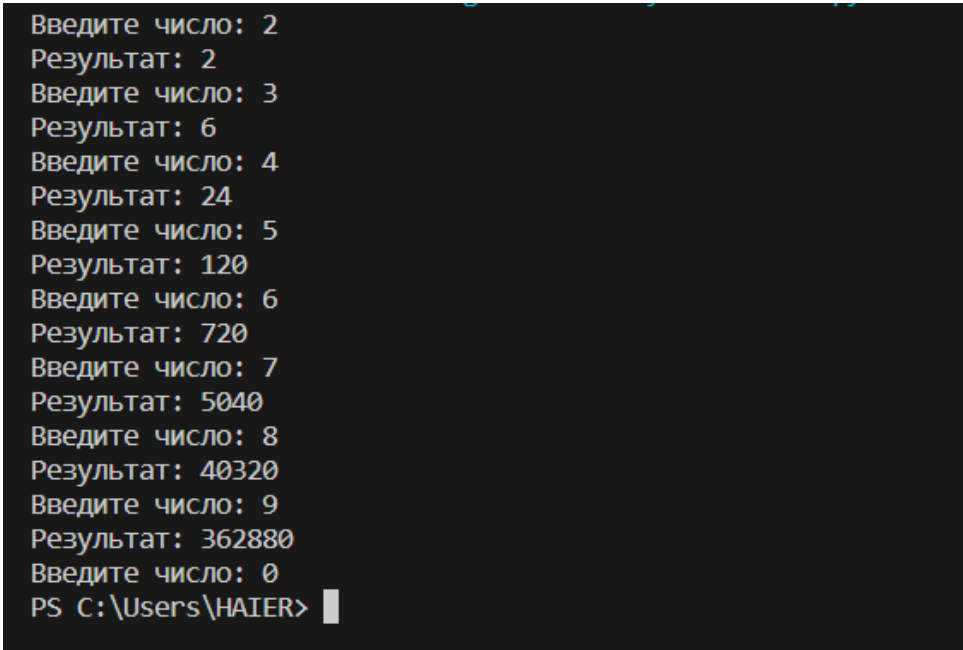
    print(f"Площадь боковой поверхности цилиндра равна: {2*pi*r*h}")
elif v == 2:
    print(f"Полная площадь цилиндра равна: {2*pi*r*h + 2*circle(r)}")

if __name__ == '__main__':
    cylinder()

```

3) 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.

Результат работы программы:



```

Введите число: 2
Результат: 2
Введите число: 3
Результат: 6
Введите число: 4
Результат: 24
Введите число: 5
Результат: 120
Введите число: 6
Результат: 720
Введите число: 7
Результат: 5040
Введите число: 8
Результат: 40320
Введите число: 9
Результат: 362880
Введите число: 0
PS C:\Users\HAIER>

```

Рисунок 5 – Результат работы программы

Код программы:

```

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

def multiple():
    S = 1
    while True:
        n = int(input("Введите число: "))
        if n == 0:
            break

    S *= n
    print(f"Результат: {S}")

```



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

4) 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.

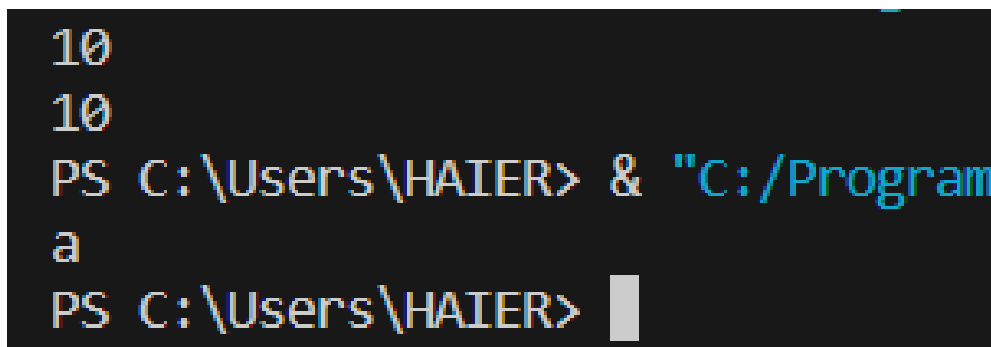
2. Функция `test_input()` имеет один параметр. В теле она проверяет, можно ли переданное ей значение преобразовать к целому числу. Если можно, возвращает логическое `True`. Если нельзя – `False`.

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.

Результат работы программы:



```
10  
10  
PS C:\Users\HAIER> & "C:/Program  
a  
PS C:\Users\HAIER>
```

Рисунок 6 – Результат работы программы

Код программы:

```
#!/usr/bin/env python3  
# -*- coding: utf-8 -*-  
  
def get_input():  
    n = input()  
    return n
```

```

def test_input(n):
    return n.isdigit()

def str_to_int(n):
    return int(n)

def print_int(n):
    print(n)

if __name__ == '__main__':
    n = get_input()
    b = test_input(n)

    if b == True:
        n_int = str_to_int(n)
        print_int(n_int)

```

3. Выполнено индивидуальное задание

Решить индивидуальное задание лабораторной работы 2.6, оформив каждую команду в виде отдельной функции.

Результат работы программы:

```

>>> help
Список команд:

add - добавить маршрут;
list - вывести список маршрутов;
select <пункт> - запросить информацию о маршруте с указанным пунктом;
help - отобразить справку;
exit - завершить работу с программой.
>>> add
Начальный пункт маршрута? Москва
Конечный пункт маршрута? Ставрополь
Номер маршрута? 1
>>> add
Начальный пункт маршрута? Ставрополь
Конечный пункт маршрута? Москва
Номер маршрута? 2
>>> list

```

№	Начальный пункт	Конечный пункт	Номер
1	Москва	Ставрополь	1
2	Ставрополь	Москва	2

```

>>> exit

```

Рисунок 7 – Результат работы программы

```

>>> add
Начальный пункт маршрута? Москва
Конечный пункт маршрута? Красноярск
Номер маршрута? 1
>>> add
Начальный пункт маршрута? Ставрополь
Конечный пункт маршрута? Москва
Номер маршрута? 2
>>> add
Начальный пункт маршрута? Краснодар
Конечный пункт маршрута? Красноярск
Номер маршрута? 3
>>> select Москва
      1: Москва-Красноярск, номер маршрута: 1
      2: Ставрополь-Москва, номер маршрута: 2
>>> █

```

Рисунок 8 – Результат работы программы

Код программы:

```

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

import sys

def add():
    """
    Добавить маршрут
    """
    name_start = input("Начальный пункт маршрута? ")
    name_end = input("Конечный пункт маршрута? ")
    number = int(input("Номер маршрута? "))

    route = {
        'name_start': name_start,
        'name_end': name_end,
        'number': number,
    }

    return route

def list(routes):
    """
    Вывести список маршрутов
    """
    if routes:
        line = '+-{}-+-{}-+-{}-+-{}-+'.format(
            '_' * 4,

```

```

        '_' * 30,
        '_' * 30,
        '_' * 8
    )
    print(line)

    print('| {:^4} | {:^30} | {:^30} | {:^8} |'.format(
        "№",
        "Начальный пункт",
        "Конечный пункт",
        "Номер"
    )
    )
    print(line)

    for idx, route in enumerate(routes, 1):
        print('| {:>4} | {:<30} | {:<30} | {:>8} |'.format(
            idx,
            route.get('name_start', ""),
            route.get('name_end', ""),
            route.get('number', 0)
        )
        )
        print(line)
    else:
        print("Список работников пуст.")

def select(routes, command):
    """
    Вывести выбранные маршруты
    """
    parts = command.split(' ', maxsplit=1)
    station = parts[1]
    count = 0

    for route in routes:
        if (station == route["name_start"].lower() or
            station == route["name_end"].lower()):

            count += 1
            print('{:>4}: {}-{}, номер маршрута: {}'.format(count,
                route["name_start"], route["name_end"], route["number"]))

    if count == 0:
        print("Маршрут не найден.")

def help():
    """
    Вывести список команд
    """

```

```

print("Список команд:\n")
print("add - добавить маршрут;")
print("list - вывести список маршрутов;")
print("select <пункт> - запросить информацию"+
      " о маршруте с указанным пунктом;")
print("help - отобразить справку;")
print("exit - завершить работу с программой.")

def main():
    """
    Основная функция
    """
    routes = []

    while True:

        command = input(">>> ").lower()

        if command == 'exit':
            break

        elif command == 'add':
            route = add()
            routes.append(route)

            if len(routes) > 1:
                routes.sort(key=lambda item: item.get('number', ''))

        elif command == 'list':
            list(routes)

        elif command.startswith('select '):
            select(routes, command)

        elif command == 'help':
            help()

        else:
            print(f"Неизвестная команда {command}", file=sys.stderr)

if __name__ == '__main__':
    main()

```

Ответы на контрольные вопросы

1. Каково назначение функций в языке программирования Python?

Функции нужны для того, чтобы делать код более компактным и читаемым.

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

С помощью оператора `def` в языке Python определяются функции.

Оператор `return` нужен для того, чтобы функция могла возвращать значение.

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

Локальные переменные видны только в локальной области видимости, которой может выступать отдельно взятая функция. Глобальные переменные видны во всей программе. Локальные переменные используются там, где нет необходимости использовать глобальную переменную, а глобальные нужны для того, чтобы можно было изменять их из локальных областей видимости.

4. Как вернуть несколько значений из функции Python?

Для того, чтобы вернуть несколько значений из функции, необходимо использовать кортежи.

5. Какие существуют способы передачи значений в функцию?

Существует передача аргументов в функцию по ссылке и по значению.

6. Как задать значение аргументов функции по умолчанию?

Для того, чтобы задать значение аргументов функции по умолчанию, необходимо присвоить им значение при определении функции.

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

`lambda`-выражения позволяют создавать анонимные функции в одну строку.

8. Как осуществляется документирование кода согласно PEP257?

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).

Все модули должны, как правило, иметь строки документации, и все функции и классы, экспортируемые модулем также должны иметь строки

документации. Публичные методы (в том числе `__init__`) также должны иметь строки документации. Пакет модулей может быть документирован в `__init__.py`.

Для согласованности, необходимо использовать `"""triple double quotes"""` для строк документации. Нужно использовать `r"""raw triple double quotes"""`, если в строке документации будет использоваться обратная косая черта.

Существует две формы строк документации: однострочная и многострочная.

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

Многострочные строки документации состоят из однострочной строки документации с последующей пустой строкой, а затем более подробным описанием.

Однострочные строки предназначены для действительно очевидных случаев. Они должны уместиться на одной строке.

Вывод

В ходе выполнения работы приобретены навыки по работе с функциями при написании программ с помощью языка программирования Python версии 3.x