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 №2.8

Дисциплины «Программирование на Python»

Выполнил:

Пустяков Андрей Сергеевич

2 курс, группа ИВТ-б-о-22-1,

09.03.01 «Информатика и вычислительная техника (профиль) «Программное обеспечение средств вычислительной техники и автоматизированных систем», очная форма обучения

_____

(подпись)

Руководитель практики:

Воронкин Р. А. кандидат технических наук, доцент, доцент кафедры инфокоммуникаций

_____

(подпись)

Ставрополь, 2023 г.

Тема: Работа с функциями в языке Python.

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

Ход работы:

Создание репозитория, клонирование репозитория, организация репозитория согласно модели ветвления «git-flow» (рис. 1).

```
C:\Users\Andrey\Desktop\пайтон\10 лаба\Python_laboratory_work_2.7>cd C:\Users\Andrey\Desktop\пайтон\11 лаба

C:\Users\Andrey\Desktop\пайтон\11 лаба>git clone https://github.com/AndreyPust/Python_laboratory_work_2.8.git
Cloning into 'Python_laboratory_work_2.8'...
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (5/5), done.

C:\Users\Andrey\Desktop\пайтон\11 лаба>cd C:\Users\Andrey\Desktop\пайтон\11 лаба\Python_laboratory_work_2.8

C:\Users\Andrey\Desktop\пайтон\11 лаба\Python_laboratory_work_2.8>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/]
```

Рисунок 1 – Организация модели ветвления «git-flow».

Проработка примеров лабораторной работы:

Пример 1.

Необходимо было оформить каждую команду из примера лабораторной работы 2.6 в виде отдельных функций (тело главной функции также оформить с помощью функции).

Код программы примера №1 и результаты работы программы (рис. 2, 3).

```python
#!/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
```

```python
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()
```

Рисунок 2 – Код программы примера №1.

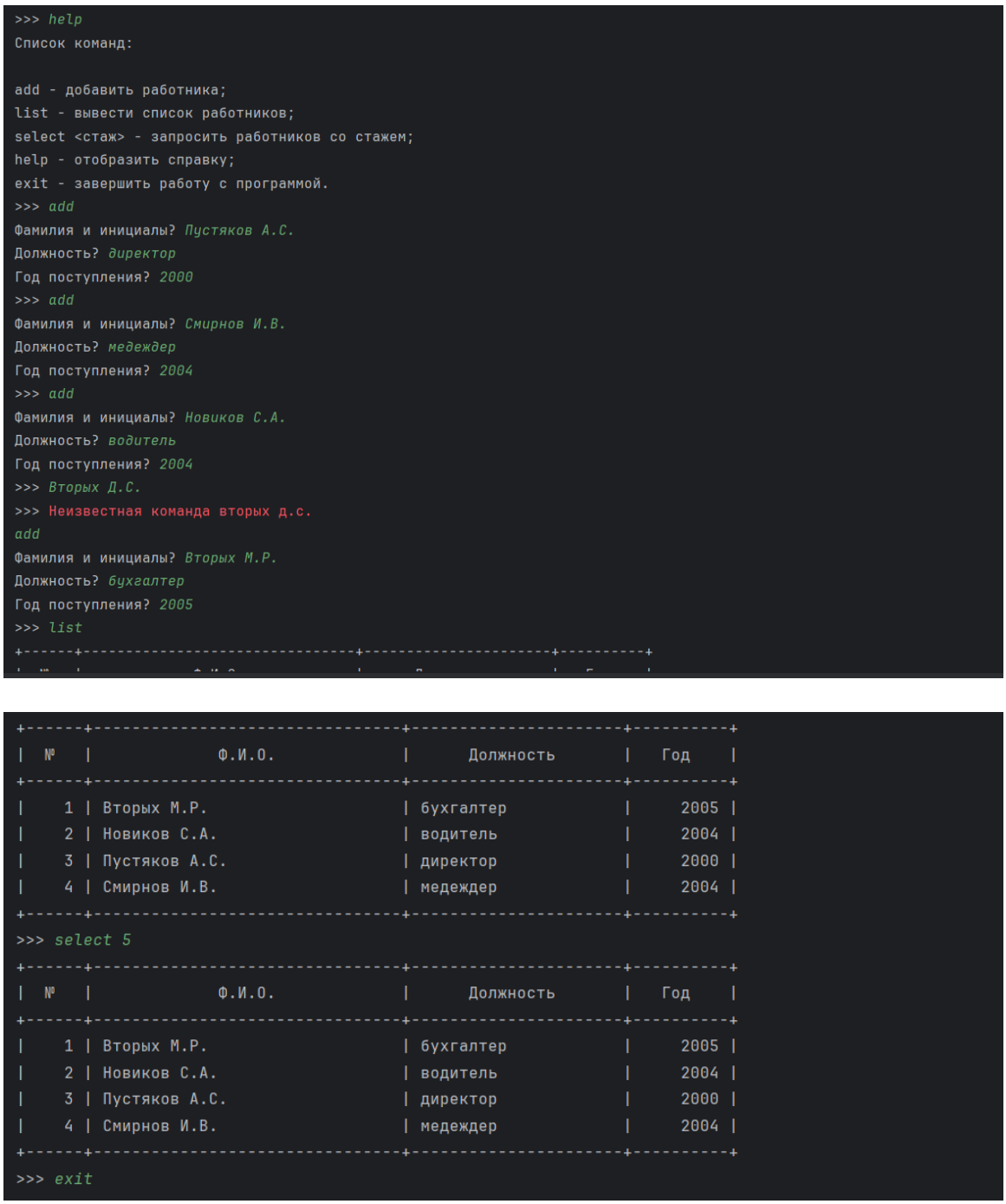

Рисунок 3 – Результаты работы программы примера №1.

Необходимо решить задачу: основная ветка программы, не считая заголовков функций, состоит из двух строк 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.

Код программы решения данной задачи и результаты работы программы при объявлении функций *positive(), negative() и test()* в разных последовательностях (рис. 4, 5, 6).

```
# -*- coding: utf-8 -*-


1 usage  new *
def test():
    """Определение числа, положительное или отрицательное."""

    number = int(input("Веведите число (полижительное или отрицательное): "))
    if number >= 0:
        positive()
    else:
        negative()


1 usage  new *
def negative():
    """Функция вывода сообщения об отрицательности числа."""

    print("Отрицательное")


1 usage  new *
def positive():
    """Функция вывода сообщения о положительности числа."""

    print("Положительное")


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

Рисунок 4 – Код программы первой задачи.

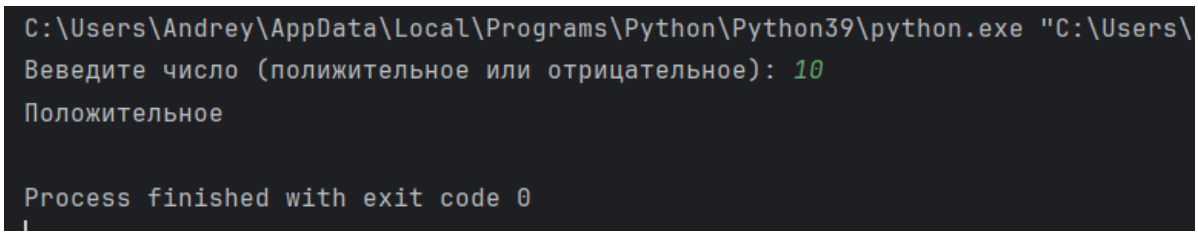

Рисунок 5 – Результаты работы программы первой задачи при обычном порядке объявления функций.

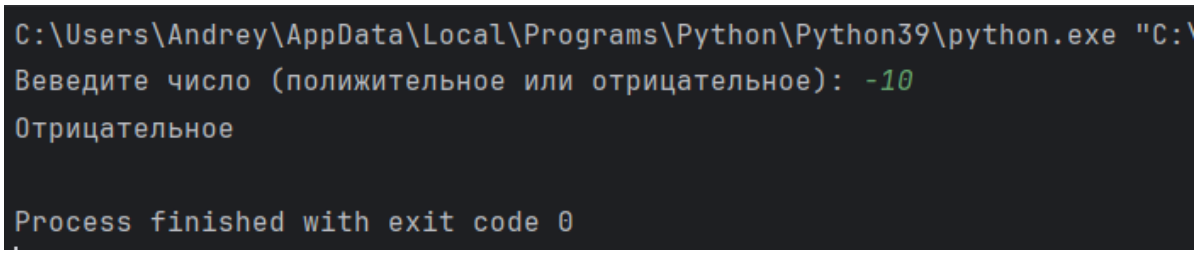

Рисунок 6 – Результаты работы программы первой задачи при обычном порядке объявления функций.

При объявлении функций в другом порядке изменений не произошло, поскольку последовательность объявления всех функций не имеет значения. Нет особой значимости обращаться к участку кода, если он расположен раньше или позже других функций. Ошибок не происходит потому, что используемые переменные в коде не используются в функциях и наоборот (в функциях используются локальные переменные). Самое главное для работы кода – это объявление функций до их вызова, а в какой последовательности были объявлены функции не важно. До тех пор, пока интерпретатор не дошел до кода программы, он объявит все функции и не одну не оставит без объявления, даже если в одной функции вызывается другая (тело функции не начнет исполняться до тех пор, пока ее не вызовут).

Необходимо решить 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()*.

Код программы решения задачи №2 и результаты работы программы в случае выбора нахождения только площади круга и в случае нахождения площади всего цилиндра (рис. 6, 7).

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

"""
Необходимо решить 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().
"""


def cylinder():
    """Функция расчета площади цилиндра"""

    def circle(radius):
        """Расчет площади круга"""

        return 3.14 * (radius ** 2)

    cucle = input("Если вы хотете вычислить получить только площадь боковой
поверхности нажмите y: ")
    radius = float(input("Введите радиус: "))
    height = float(input("Введите высоту:  "))
    answer = 2 * 3.14 * radius * height

    """
    Расчет требуемой площади. Если нужна всего цилиндра - то к результату
    добавляются две площади круга.
    """

    if cucle == 'y':
        print("Площадь круга: ", answer)
    else:
        print("Площадь цилиндра: ", answer + 2 * circle(radius))


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

Рисунок 6 – Код программы второй задачи.

```
Если вы хотете вычислить получить только площадь боковой поверхности нажмите y: y
Введите радиус: 10
Введите высоту:  10
Площадь круга:  628.0
```

```
C:\Users\Andrey\AppData\Local\Programs\Python\Python39\python.exe "C:\Users\Andrey
Если вы хотете вычислить получить только площадь боковой поверхности нажмите y: n
Введите радиус: 10
Введите высоту:  10
Площадь цилиндра:  1256.0
```

Рисунок 7 – Результаты работы программы второй задачи.

Необходимо решить задачу: написать функцию, которая считывает с клавиатуры числа и перемножает их до тех пор, пока не будет введен 0. Функция должна возвращать полученное произведение. Вызвать функцию и вывести на экран результат ее работы.

Код программы решения задачи №3 и результаты работы программы (рис. 8, 9).

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


def multiply():
    new_value = int(input("Введите число: "))
    if new_value == 0:
        return 0
    answer = 1

    """Ввод чисел до тех пор, пока не будет введент 0 пользователем."""

    while new_value != 0:
        answer *= new_value
        new_value = int(input("Введите следующее число: "))
    return answer


if __name__ == '__main__':

    """Перемножение всех введённых пользователем чисел (кроме нуля)."""

    print("Результат перемножения всех введенных чисел с клавиатуры: ",
multiply())
```

Рисунок 8 – Код программы третьей задачи.

```
C:\Users\Andrey\AppData\Local\Programs\Python\Python39\python.exe "C:\Users\Andre
Введите число: 1
Введите следующее число: 2
Введите следующее число: 3
Введите следующее число: 4
Введите следующее число: 5
Введите следующее число: 6
Введите следующее число: 7
Введите следующее число: 8
Введите следующее число: 9
Введите следующее число: 0
Результат перемножения всех введенных чисел с клавиатуры:  362880
```

Рисунок 9 – Результаты работы программы третьей задачи.

Необходимо решить следующую задачу: необходимо создать четыре функции: «get_input()» не имеет параметров, запрашивает ввод с клавиатуры и возвращает в основную программу полученную строку, «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, «str_to_int()» имеет один параметр, в теле преобразовывает переданное значение к целочисленному типу, возвращает полученное число, «print_int()» имеет один параметр, она выводит переданное значение на экран и ничего не возвращает.

Код программы решения задачи №4 и результаты работы программы в случае, когда строку можно привести к целочисленному типу и когда нельзя (рис. 10, 11).

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

"""
Программа содержит четыре функции:
 - «get_input()» не имеет параметров, запрашивает ввод с клавиатуры и
возвращает в основную
 программу полученную строку;
 - «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;
 - «str_to_int()» имеет один параметр, в теле преобразовывает переданное
значение к целочисленному
 типу, возвращает полученное число;
 - «print_int()» имеет один параметр, она выводит переданное значение на
экран и ничего не возвращает.
"""
```

```python
def get_input():
    """
    Запрос ввода строки с клавиатуры.
    """
    return input("Введите строку: ")


def test_input(word):
    """
    Анализ строки, можно ли преобразовать ее в число.
    """
    if word.isdigit():
        return True
    else:
        return False


def str_to_int(value):
    """
    Преобразование строки к целочисленному типу.
    """
    return int(value)


def print_int(value):
    """
    Вывод числа на экран.
    """
    print(value)


if __name__ == '__main__':
    """
    Тело главной функции, которая последовательно вызывает все объявленные
ранее функции для работы со строкой.
    """
    value = get_input()
    if test_input(value):
        print_int(str_to_int(value))
```

Рисунок 10 – Код программы четвертой задачи.

```
C:\Users\Andrey\AppData\Local\Programs\Python\Python39\python.exe "C:\Users\Andrey\
Введите строку: Кирпич ни с того ни с сего никому и никогда на голову не свалится.

Process finished with exit code 0
```

```
C:\Users\Andrey\AppData\Local\Programs\Python\Python39\python.exe "C:\U
Введите строку: 12345678910
12345678910
```

Рисунок 11 – Результаты работы программы третьей задачи.

Выполнение индивидуального задания:

Необходимо оформить решение индивидуального задания из лабораторной работы 2.6 в виде функций (оформить каждую команду в виде функций) (Вариант 26 (7)).

Код программы индивидуального задания и результаты работы программы (рис. 12, 13).

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

"""
Необходимо использовать словарь, содержащий следующие ключи: название пункта
назначения, номер поезда, время отправления. Написать программу, выполняющую
следующие действия: ввод с клавиатуры данных в список, состоящий из словарей
заданной структуры; записи должны быть упорядочены по времени отправления
поезда;
вывод на экран информации о поездах, направляющихся в пункт, название
которого
введено с клавиатуры; если таких поездов нет, выдать на дисплей
соответствующее сообщение.
Необходимо оформить команды в виде отдельных функций.
"""

import sys


def get_station(stations):
    """
    Функция запроса данных о станции.
    """
    name = input("Название пункта: ")
    # Создать словарь.
    station = {'name': name}
    print("Добавить поезд? n/y")
    cucle = input()
    if cucle == 'n':
        station['train'] = 'Поездов нет'
        station['time'] = ' '
    else:
        train = input("Номер поезда: ")
        dep_time = input("Время отправления поезда: ")
        # Добавить в словарь поезд и время.
        station['train'] = train
        station['time'] = dep_time

    # Добавить словарь в список.
    stations.append(station)

    # Отсортировать список в случае необходимости по времени поезда.
    if len(stations) > 1:
        stations.sort(key=lambda item: item.get('time', ''))

    return stations


def info(stations, name_station):
```

```
    """
    Функция вывода информации о введенной станции, о поездах и их времени
(если они есть).
    Функция ничего не возвращает.
    """
    count = 0
    for station in stations:
        if station.get('name') == name_station:
            count += 1
            print("Номер поезда пункта: ", station.get('train'),
                  "Время отправления: ", station.get('time'))

    # Если счетчик равен 0, то станции не найдены.
    if count == 0:
        print("Станции не найдены.")

    return None


def help_command():
    """
    Функция вывода справочной информации о доступных командах, ничего не
возвращает.
    """
    print("Список команд:\n")
    print("add - добавить станцию;")
    print("info <станция> - запросить информацию о станции;")
    print("help - отобразить справку;")
    print("exit - завершить работу с программой.")

    return None


def main():
    """
    Главная функция программы.
    """
    # Список пунктов (станций).
    stations = []

    # 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':
            get_station(stations)

        elif command.startswith('info '):
            # Разбить команду на части для выделения названия пункта.
            name_station = command.split(' ', maxsplit=1)
            name_station = name_station[1]
            # Вызвать функцию вывода информации о станции.
            info(stations, name_station)

        elif command == 'help':
            help_command()

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

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

Рисунок 12 – Код программы индивидуального задания.

```
C:\Users\Andrey\AppData\Local\Programs\Python\Python39\python.exe "C:\Users\Andrey\D
>>> help
Список команд:

add - добавить станцию;
info <станция> - запросить информацию о станции;
help - отобразить справку;
exit - завершить работу с программой.
>>> add
Название пункта: первинск
Добавить поезд? n/y
y
Номер поезда: 001
Время отправления поезда: 10:00
>>> add
Название пункта: второвинск
Добавить поезд? n/y
y
Номер поезда: 002
Время отправления поезда: 00:00
>>> add
Название пункта: третьинск
Добавить поезд? n/y
y
Номер поезда: 003
Время отправления поезда: 12:00
>>> info первинск
Номер поезда пункта:  001 Время отправления:  10:00
>>> exit
```

Рисунок 11 – Результаты работы программы индивидуального задания.

Ответы на контрольные вопросы:

1. Функция – это обособленный участок кода, который можно вызывать, обратившись к нему по имени, которым он был назван. При вызове происходит выполнение команд тела функции. Функции можно сравнить с

2. небольшими программками, которые сами по себе, т. е. автономно, не исполняются, а встраиваются в обычную программу. Других ключевых отличий функций от программ нет. Функции также при необходимости могут

получать и возвращать данные (обычно они их получают не с ввода (клавиатуры, файла и др.), а из вызывающей программы. Сюда же они возвращают результат своей работы). Существует множество встроенных в язык программирования функций (print(), input(), int(), float(), str(), type()). Программист всегда может определять свои функции (это пользовательские функции).

3. В Python оператор «def» используется для определения функций, а «return» для выхода из функции и передачи данных в то место, откуда она была вызвана.

4. Локальные переменные в Python создаются внутри функции и видны только внутри этой функции (используются для временного хранения данных внутри функции). Глобальные переменные создаются вне функций и могут быть использованы в любом месте программы. Однако, если нужно изменить значение глобальной переменной внутри функции, необходимо использовать ключевое слово «global».

5. В Python функция может вернуть несколько значений, используя кортеж (tuple). Для этого необходимо указать значения через запятую после «return» или заключить значения в фигурные скобки, что создаст кортеж.

6. Значения могут быть переданы в функцию в Python через позиционные аргументы (передаются в порядке, в котором они определены в определении функции), ключевые аргументы (передаются с указанием имени параметра) и аргументы по умолчанию (значения уже установлены по умолчанию => могут быть пропущены при вызове функции).

7. В Python значения аргументов функции можно задать по умолчанию, что позволяет вызывать функцию без указания значений для этих аргументов, если они необходимы. Задание значений по умолчанию осуществляется в определении функции, например: def function(example, example_two = 555, example_thee = "smth_else").

8. «lambda» функции в Python – такие функции, которые не имеют названия. Их также называют анонимными. Слово «lambda» является

служебным, и не отражает сути конкретной функции. Основная причина применения лямбда функций — это создание функции, которая используется в коде единожды.

9. PEP 257 содержит рекомендации по документированию строк (docstrings) в коде на языке Python: документационные строки следует писать в тройных кавычках ("""smth"""); для модуля, класса или функции они должны быть расположены в начале их определения; для функций строки должны включать описание параметров, возвращаемого значения и возможных исключений.

10. Строки документации начинаются и заканчиваются тремя кавычками, однако однострочная форма занимает не больше одной строки и предназначена для кратких описаний, в то время как многострочная форма занимает несколько строк и предоставляет более развёрнутое описание.

Вывод: в ходе лабораторной работы была подробно изучена работа с функциями в Python, были изучены операторы объявления функции, правила и порядок объявлений функций в коде, были изучены понятия локальных и глобальных переменных, были изучены способы передачи переменных функции и способы возвращения переменных из функций. Также были изучены правила документирования кода и функций согласно PEP 256.