

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 «Программирование на Python»**  
**Вариант \_\_\_\_**

Выполнил:  
Иващенко Олег Андреевич  
2 курс, группа ИВТ-б-о-22-1,  
09.03.02 «Информационные и  
вычислительные машины»,  
направленность (профиль)  
«Программное обеспечение средств  
вычислительной техники и  
автоматизированных систем»

---

(подпись)

Руководитель практики:  
Воронкин Роман Александрович,  
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

Таблица 1 – Код программы example.py

```
#!/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()

```

```

>>> add
Фамилия и инициалы? Иванов И.И.
Должность? -
Год поступления? 2020
>>> list
+-----+-----+-----+-----+
| № |          Ф.И.О.          | Должность |      Год      |
+-----+-----+-----+-----+
|  1 | Иванов И.И.              | -         |      2020     |
+-----+-----+-----+-----+

```

Рисунок 1 – Вывод программы example.py

Задание 1. Основная ветка программы, не считая заголовков функций, состоит из двух строк кода. Это вызов функции test() и инструкция 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(), её тело содержит выражение вывода на экран слова «Отрицательное».

Таблица 2 – Код программы general\_1.py

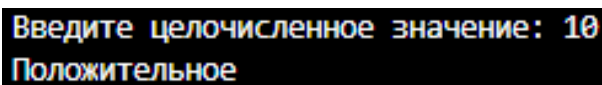
```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

def test(a):
    if a >= 0:
        positive()
    else:
        negative()

def positive():
    print("Положительное")

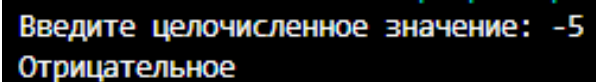
def negative():
    print("Отрицательное")

if __name__ == "__main__":
    test(int(input("Введите целочисленное значение: ")))
```

A screenshot of a terminal window showing the output of the program. The prompt "Введите целочисленное значение: " is followed by the user input "10". The program then outputs "Положительное".

Введите целочисленное значение: 10  
Положительное

Рисунок 2.1 – Вывод программы general\_1.py при вводе положительного значения

A screenshot of a terminal window showing the output of the program. The prompt "Введите целочисленное значение: " is followed by the user input "-5". The program then outputs "Отрицательное".

Введите целочисленное значение: -5  
Отрицательное

Рисунок 2.2 – Вывод программы general\_2.py при вводе отрицательного значения

Задание 2. В основной ветке программы вызывается функция `cylinder()`, которая вычисляет площадь цилиндра. В теле `cylinder()` определена функция `circle()`, вычисляющая площадь круга по формуле  $\pi r^2$ . 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()`.

Таблица 3 – Код программы `general_2.py`

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

import math

def cylinder(r, h):
    """
    Вычисление площади цилиндра
    """

    side = 2 * math.pi * r * h
    answer = input("Хотите ли Вы получить полную площадь цилиндра? (y/n) ")
    if answer == 'y':
        def circle(r):
            square = math.pi * r**2
            return square

        return circle(r) + side * 2
    else:
        return side

if __name__ == "__main__":
    """
    Основная программа
    """

    print(cylinder(float(input("Введите радиус: ")),
                    float(input("Введите высоту: "))))
```

```
Введите радиус: 5
Введите высоту: 10
Хотите ли Вы получить полную площадь цилиндра? (y/n) n
314.1592653589793
```

Рисунок 3.1 – Вывод программы general\_2.py при отказе от вывода полной площади

```
Введите радиус: 5
Введите высоту: 10
Хотите ли Вы получить полную площадь цилиндра? (y/n) y
706.8583470577034
```

Рисунок 3.2 – Вывод программы general\_2.py при согласии на вывод полной площади

Задание 3. Написать 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 её работы.

Таблица 4 – Код программы general\_3.py

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

def multiply():
    """
    Функция перемножения двух чисел
    """
    a = int(input("Введите первое число: "))
    if a != 0:
        b = int(input("Введите второе число: "))
        if b != 0:
            print(f"{a} * {b} = {a * b}")
            multiply()
        else:
            print("Завершение работы программы...")

if __name__ == "__main__":
    """
    Основная программа
    """
```

```
multiply()
```

```
Введите первое число: 5
Введите второе число: 10
5 * 10 = 50
Введите первое число: 25
Введите второе число: 84
25 * 84 = 2100
Введите первое число: 0
Завершение работы программы...
```

Рисунок 4 – Результат работы программы general\_3.py

Задание 4. Написать программу, в которой определены следующие четыре функции:

- Функция `get_input()` не имеет параметров, запрашивает ввод с клавиатуры и возвращает в основную программу полученную строку.
- 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`.
- Функция `str_to_int()` имеет один параметр. В теле преобразовывает переданное значение к целочисленному типу. Возвращает полученное число.
- Функция `print_int()` имеет один параметр. Она выводит переданное значение на экран и ничего не возвращает.

В основной ветке программы вызывается первая функция. То, что она вернула, передаётся во вторую функцию. Если вторая функция вернула `True`, то те же данные (из первой функции) передаются в третью функцию, а возвращённое третьей функцией значение – в четвёртую.

Таблица 5 – Код программы general\_4.py

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



```
def get_input():
    """
    Функция запрашивает ввод с клавиатуры и возвращает
    полученную строку
    """

    string = input("Введите строку: ")
    return string

def test_input(string):
    """
    Функция проверяет, возможно ли преобразовать полученную строку
    в целочисленное значение. Если можно, то возвращает логическое True,
    если нельзя - логическое False
    """

    if string.isdigit():
        return True
    else:
        return False

def str_to_int(string):
    """
    Функция преобразует полученную строку в целочисленное значение
    и возвращает его
    """

    integer = int(string)
    return integer

def print_int(integer):
    """
    Функция выводит полученное целочисленное значение на экран
    """

    print(f"Полученное число: {integer}")

if __name__ == "__main__":
    string = get_input()
    if test_input(string) == True:
        print_int(str_to_int(string))
    else:
        print("Введённое число не удалось конвертировать. Завершение работы")
```

```
Введите строку: 2567
Полученное число: 2567
```

Рисунок 5.1 – Вывод программы general\_4.py при вводе числового значения

```
Введите строку: 76s54a
Введённое число не удалось конвертировать. Завершение работы
```

Рисунок 5.2 – Вывод программы general\_4.py при вводе не числового значения

Индивидуальное задание.

Таблица 6 – Код программы individual.py

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

def print_help():
    """
    Функция вывода доступных пользователю команд
    """

    print("list - вывод всех добавленных записей")
    print("add - добавление новых записей")
    print("find - найти запись по фамилии")
    print("exit - завершение работы программы")

def add():
    """
    Функция добавления новой записи, возвращает запись
    """

    surname = input("Введите фамилию: ")
    name = input("Введите имя: ")
    phone = input("Введите номер телефона: ")
    date = tuple(map(int, input("Введите дату рождения: ").split('.')))

    new_member = {'surname': surname,
                  'name': name,
                  'phone': phone,
                  'date': date
                 }

    return new_member
```

```

def print_list(list):
    """
    Функция выводит на экран список всех существующих записей
    """

    for member in member_list:
        print(f"{member['surname']} {member['name']}, "
              f"{member['phone']}, {member['date']}")

def find_member(surname):
    """
    Функция для вывода на экран всех записей, чьи фамилии совпадают
    с введённой (не возвращает никаких значений)
    """

    count = 0

    for member in member_list:
        if member['surname'] == surname:
            print(f"{member['surname']} {member['name']}, "
                  f"{member['phone']}, {member['date']}")
            count += 1

    if count == 0:
        print("Записи не найдены")

if __name__ == "__main__":
    """
    Основная программа
    """

    member_list = []

    while True:
        cmd = input(">>> ")

        if cmd == "help":
            print_help()

        elif cmd == "add":
            member_list.append(add())
            member_list.sort(key=lambda item: item.get('phone')[:3])

        elif cmd == "list":
            print_list(member_list)

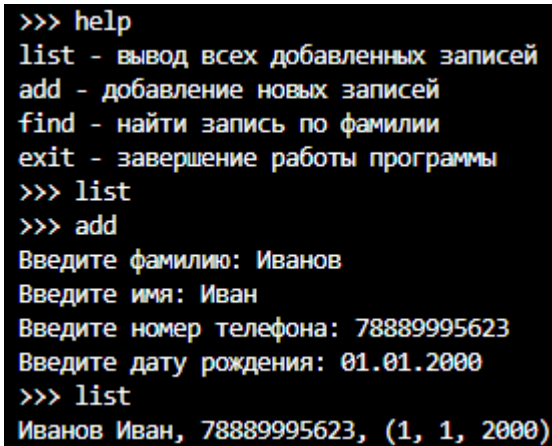
        elif cmd == "find":
            find_member(input("Введите фамилию: "))

        elif cmd == "exit":

```

```
print("Завершение работы программы...")
break

else:
    print(f"Команды {cmd} не существует")
```



```
>>> help
list - вывод всех добавленных записей
add - добавление новых записей
find - найти запись по фамилии
exit - завершение работы программы
>>> list
>>> add
Введите фамилию: Иванов
Введите имя: Иван
Введите номер телефона: 78889995623
Введите дату рождения: 01.01.2000
>>> list
Иванов Иван, 78889995623, (1, 1, 2000)
```

Рисунок 6 – Результат работы программы individual.py

### Контрольные вопросы

1. Каково назначение функций в языке программирования Python?

Функция в программировании представляет собой обособленный участок кода, который можно вызывать, обратившись к нему по имени, которым он был назван. При вызове происходит выполнение команд тела функции.

2. Каково назначение операторов def и return?

С помощью оператора def в языке программирования Python определяют функции. Пример:

```
def countFood():
    a = int(input())
    b = int(input())
    print("Всего ", a + b, "шт.")
```

С помощью оператора return вызываемая функция может передавать какие-либо данные из своих тел в основную ветку программу, откуда данная функция была вызвана.

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

Локальные переменные видны только в локальной области, которой может выступать отдельно взятая функция.

Глобальные переменные видны (доступны) во всей программе.

К глобальной переменной можно обратиться из локальной области видимости. К локальной переменной нельзя обратиться из глобальной области видимости, потому что локальная переменная существует только в момент выполнения тела функции. При выходе из неё локальная переменная исчезает, а компьютерная память, выделенная для неё, освобождается.

4. Как вернуть несколько значений из функции Python?

В Python позволительно возвращать из функции несколько объектов, перечислив их через запятую после команды return. Пример:

```
def cylinder():  
    r = float(input())  
    h = float(input())  
    side = 2 * math.pi * r * h  
    circle = math.pi * r**2  
    full = side + 2 * circle  
    return side, full
```

5. Какие существуют способы передачи значений в функцию?

Функции могут принимать данные, что реализуется с помощью «параметров», которые указываются в скобках в заголовке функции. Количество параметров может быть любым.

Параметры представляют собой локальные переменные, которым присваиваются значения в момент вызова функции. Конкретные значения, которые передаются в функцию при её вызове, будут называться аргументами.

6. Как задать значение аргумента функции по умолчанию?

В Python у функция бывают параметры, которым уже присвоено значение по умолчанию. В таком случае, при вызове можно не передавать соответствующие этим параметрам аргументы. Хотя можно и передать. Тогда значение по умолчанию заменится на переданное. Пример:

```
def cylinder(h, r = 1):  
    ...
```

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

Python поддерживает интересный синтаксис, позволяющий определять небольшие однострочные функции на лету. Так называемые lambda-функции могут быть использованы везде, где требуется функция. Например:

```
def func(x, y):  
    return x**2 + y**2
```

```
func = lambda x, y: x**2 + y**2
```

Принципиальные отличия def и lambda:

lambda – это выражение, а не инструкция. По этой причине ключевое слово lambda может появляться там, где синтаксис Python не позволяет использовать инструкцию def – внутри литералов или в вызовах функций.

#### 8. Как осуществляется документирование кода согласно PEP257?

Документирование кода в Python – достаточно важный аспект, ведь от неё порой зависит читаемость и быстрота понимания кода как другими людьми, так и самими автором спустя длительный промежуток времени. 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, как это написать. Этот PEP описывает соглашения, а не правила или синтаксис.

Строки документации – строковые литералы, которые являются первым оператором в модуле, функции, классе или определении метода. Такая строка документации становится специальным атрибутом `__doc__` этого объекта.

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

Для согласованности, всегда следует использовать `“””triple double quotes”””` для строк документации.

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

Однострочные строки документации предназначены для действительно очевидных случаев. Они должны уместиться в одной строке. Используются тройные кавычки, даже если документации уместается в одной строке. Позже её будет проще дополнить. Однострочная строка документации не должна быть «подписью» параметров функции/метода.

Многострочные строки документации состоят из однострочной строки документации с последующей пустой строкой, а затем более подробным описанием. Первая строка может быть использована автоматически средствами индексации, поэтому важно, чтобы она находилась на одной строке и была отделена от остальной документации пустой строкой. Первая строка может быть на той же строке, где и открываются кавычки, или на следующей строке. Вся документация должна иметь такой отступ, как и кавычки на первой строке. Вставляется пустая строка до и после всех строк документации, которые документируют класс.

**Выводы:** В процессе выполнения лабораторной работы были приобретены навыки по работе с функциями при написании программ с

помощью языка программирования Python версии 3.x, были написаны 5 программ, из которых 1 пример, 3 общих задания и 1 индивидуальное.