

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ  
ФЕДЕРАЦИИ**

**Федеральное государственное автономное  
образовательное учреждение высшего образования  
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ  
УНИВЕРСИТЕТ»**

**Кафедра инфокоммуникаций**

**«Работа с функциями в языке Python»**

**Отчет по лабораторной работе № 2.8**

**по дисциплине «Основы программной инженерии»**

Выполнил студент группы ПИЖ-б-о-21-1

Гасанов Г. М. « » 2022г.

Подпись студента\_\_\_\_\_

Работа защищена « »\_\_\_\_\_2022г.

Проверил Воронкин Р.А. \_\_\_\_\_

(подпись)

Ставрополь 2022

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

**Выполнение работы:**

1. Изучить теоретический материал работы.
2. Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT и язык программирования Python.
3. Выполните клонирование созданного репозитория.
4. Дополните файл .gitignore необходимыми правилами для работы с IDE PyCharm.
5. Организуйте свой репозиторий в соответствии с моделью ветвления git-flow.

```
[(base) svetik@MacBook-Air-Svetik LR_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/]
Version tag prefix? []
Hooks and filters directory? [/Users/svetik/Desktop/Laba11/LR_2.8/.git/hooks]
[(base) svetik@MacBook-Air-Svetik LR_2.8 %
```

Рисунок 3 – Организация репозитория в соответствии с моделью git-flow 6.

Создайте проект PyCharm в папке репозитория.

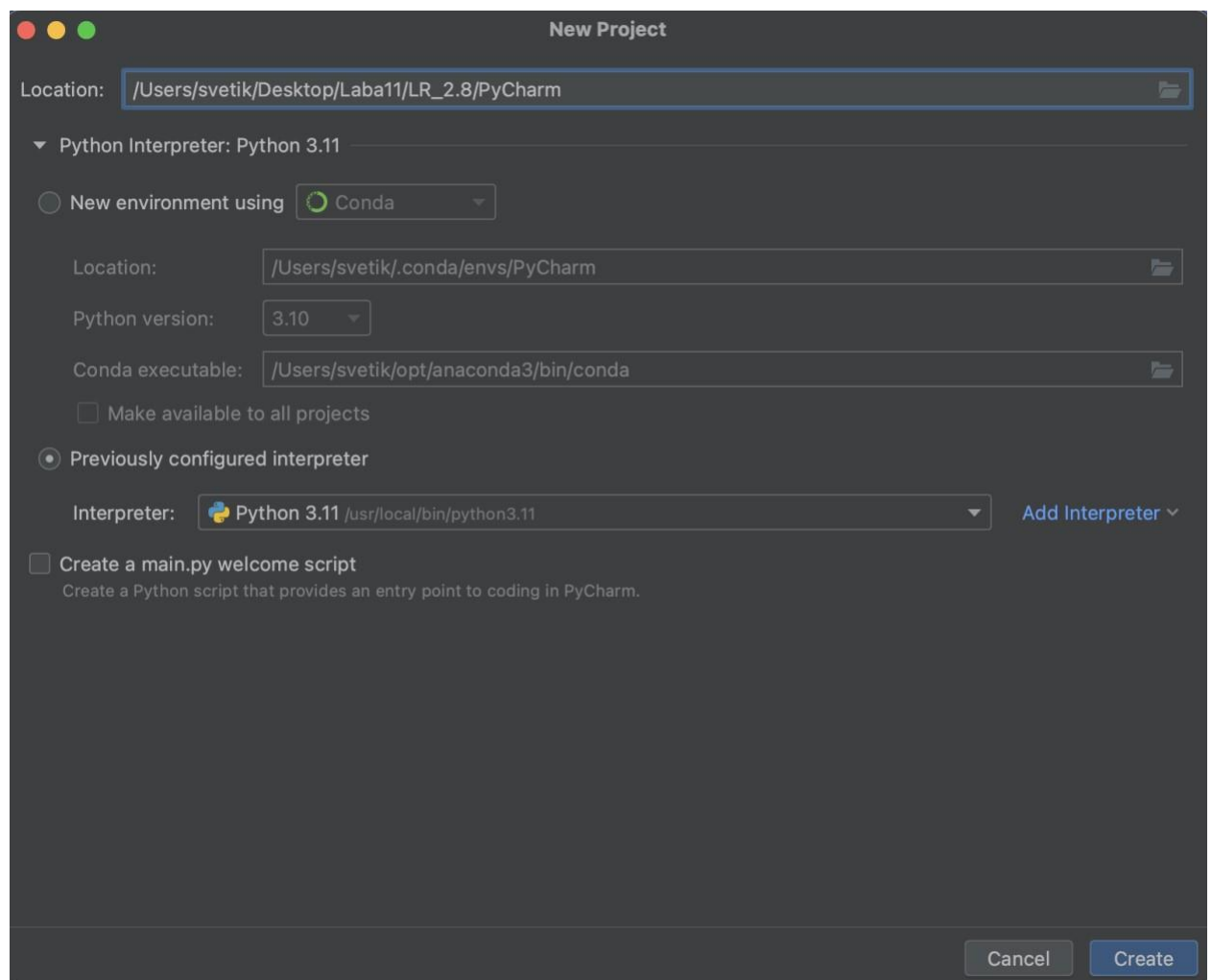


Рисунок 4 – Создание проекта PyCharm в папке репозитория

7. Проработайте примеры лабораторной работы. Зафиксируйте изменения в репозитории.

Пример 1. Определить результат выполнения операций над множествами. Считать элементы множества строками.

```

#!/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? Pupkin
Должность? Teacher
Год поступления? 2000
>>> list
+-----+-----+-----+-----+
| № |           Ф.И.О.           | Должность | Год |
+-----+-----+-----+-----+
|  1 | Pupkin                     | Teacher   | 2000 |
+-----+-----+-----+-----+
>>> 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 5 – Результат работы программы

8. Решите 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".

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

def test(number):
    if number > 0:
        positive()

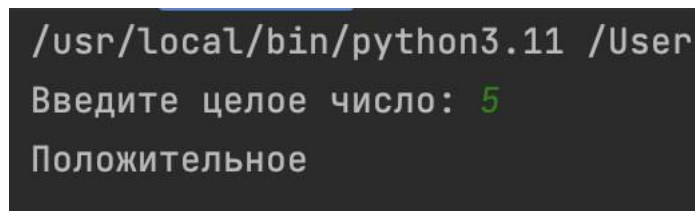
    elif number < 0:
        negative()

    else:
        print("Ноль")

def positive():
    print("Положительное")

def negative():
    print("Отрицательное")

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



```
/usr/local/bin/python3.11 /User
Введите целое число: 5
Положительное
```

Рисунок 6 – Результат работы программы

9. Зафиксируйте сделанные изменения в репозитории.

10. Решите 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()`.



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

import math

def cylinder(r, h):
    y_n = input("Нужна полная площадь цилиндра? да\нет: ").lower()

    if y_n == 'да':
        S_b = 2 * math.pi * r * h
        S = S_b + 2 * circle(r)
        print(S)

    else:
        print(2 * math.pi * r * h)

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

if __name__ == '__main__':
    r = int(input("r = "))
    h = int(input("h = "))
    cylinder(r, h)
```

```
r = 6
h = 2
Нужна полная площадь цилиндра? да\нет: да
301.59289474462014

Process finished with exit code 0
```

Рисунок 7 – Результат работы программы

11. Зафиксируйте сделанные изменения в репозитории.

12. 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.

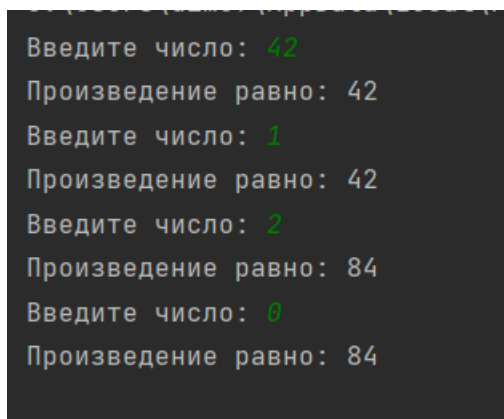
```

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

def counter():
    n = 1
    while True:
        x = int(input("Введите число: "))
        if x == 0:
            return print(f"Произведение равно: {n}")
        else:
            n *= x
            print(f"Произведение равно: {n}")

if __name__ == '__main__':
    counter()

```



```

Введите число: 42
Произведение равно: 42
Введите число: 1
Произведение равно: 42
Введите число: 2
Произведение равно: 84
Введите число: 0
Произведение равно: 84

```

Рисунок 8 – Результат работы программы

13. Зафиксируйте изменения в репозитории.

14. 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 -*-
import sys

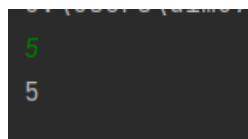
def get_input():
    return input()

def test_input(x):
    return x.isnumeric()

def str_to_int(x):
    return int(x)

def print_int(x):
    print(x)

if __name__ == '__main__':
    if test_input(x := get_input()):
        print_int(str_to_int(x))
    else:
        print("Введено не целое число", file=sys.stderr)
```



5  
5

Рисунок 9 – Результат работы программы

15. Зафиксируйте изменения в репозитории.

16. Приведите в отчете скриншоты работы программ решения индивидуального задания.

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

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

import sys
def
get_way():
    """
    Запросить данные о маршруте.
    """
    start = input('Название начального маршрута: ')
    finish = input('Название конечного маршрута: ')
    num = int(input('Номер маршрута: '))

    # Вернуть словарь.
    return {
        'start': start,
        'finish': finish,
        'num': num
    }
def
display_way(numbers):
```

```

        """
        Отобразить список маршрутов.
        """
        if
numbers:
        # Заголовок таблицы.
        line =
        '+-{}-+-{}-+-{}-+-{}-+'.format(
            '-' * 4,
            '-' * 30,
            '-' * 30,
            '-' * 15
        )
        print(line)
        print(
            '| {:^4} | {:^30} | {:^30} | {:^15} |'.format(
                "No",
                "Название начального маршрута",
                "Название конечного маршрута",
                "Номер маршрута"
            )
        )
        print(line)

        # Вывести данные о всех маршрутах.
        for idx, way in enumerate(numbers, 1):
            print(
                '| {:>4} | {:<30} | {:<30} | {:>15} |'.format(
                    idx,
                    way.get('start', ''),
                    way.get('finish', ''),
                    way.get('num', 0)
                )
            )
            print(line)

        else:
            print("Список пуст.")
            def find_way(numbers,
nw):
                """
                Выбрать маршрут с данным номером.
                """
                # Список маршрутов
                result
= []
                for h in numbers:
                    if nw in str(h.values()):
                        result.append(h)

            # Проверка на наличие записей
            if len(result) == 0:
                return print("Запись не найдена")

            # Возвратить список выбранных работников.
            return result
            def
main():
                """
                Главная функция программы.
                """
                #
Маршруты
ways = []

```

```
# 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.
            way = get_way()

            # Добавить словарь в список.
            ways.append(way)
            # Отсортировать список в случае необходимости.
            if len(ways) > 1:
                ways.sort(key=lambda item: item.get('num', 0))
        elif command == 'list':
            # Отобразить все маршруты.
            display_way(ways)
        elif command ==
'find':
            f = input('Введите номер маршрута: ')
            selected = find_way(ways, f)
            display_way(selected)
        elif command ==
'help':
            # Вывести справку о работе с программой.
            print("Список команд:\n")
            print("add -
добавить маршрут;")
            print("list - вывести
список маршрутов;")
            print("find - вывод
информации о маршруте;")
            print("help -
отобразить справку;")
            print("exit - завершить работу с программой.")
        else:
            print(f"Неизвестная команда {command}", file=sys.stderr)

    if __name__ ==
'__main__':
        main()

```

```

add
Название начального маршрута: Pushkina
Название конечного маршрута: Ne Pushkina
Номер маршрута: 34
>>> add
Название начального маршрута: Lenina
Название конечного маршрута: Ne Lenina
Номер маршрута: 45
>>> list
+-----+-----+-----+-----+-----+
| No | Название начального маршрута | Название конечного маршрута | Номер маршрута |
+-----+-----+-----+-----+-----+
| 1 | Pushkina | Ne Pushkina | 34 |
| 2 | Lenina | Ne Lenina | 45 |
+-----+-----+-----+-----+
>>> help
Список команд:

add - добавить маршрут;
list - вывести список маршрутов;
find - вывод информации о маршруте;
help - отобразить справку;
exit - завершить работу с программой.
>>>

```

Рисунок 10 – Результат работы программы

## Вопросы для защиты работы

### 1. Каково назначение функций в языке программирования Python?

Функция – это средство (способ) группирования фрагментов программного кода таким образом, что этот программный код может вызываться многократно с помощью использования имени функции. Использование функций в программах на Python даёт следующие взаимосвязанные преимущества: избежание повторения одинаковых фрагментов кода в разных частях программы; уменьшение избыточности исходного кода программы. Как следствие, уменьшение логических ошибок программирования; улучшенное восприятие исходного кода программы в случаях, где вместо блоков многочисленных инструкций (операторов) вызываются имена готовых протестированных функций. Это, в свою очередь, также уменьшает количество ошибок; упрощение внесения изменений в повторяемых блоках кода, организованных в виде функций. Достаточно внести изменения только в тело функции, тогда во всех вызовах данной функции эти изменения будут учтены; с помощью функций удобно разбивать сложную систему на более простые части. Значит, функции – удобный



способ структурирования программы; уменьшение трудозатрат на программирование, а, значит, повышение производительности работы программиста.

## **2. Каково назначение операторов def и return?**

Оператор def, выполняемый внутри определения функции, определяет локальную функцию, которая может быть возвращена или передана. Свободные переменные, используемые во вложенной функции, могут обращаться к локальным переменным функции, содержащей def.

Оператор return [выражение] возвращает результат из функции.

Оператор return без аргументов аналогичен return None

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

Области видимости определяют, в какой части программы мы можем работать с той или иной переменной, а от каких переменная «скрыта».

Так глобальные переменные доступны в любой точке программы, а локальные переменные, только в функциях, где они объявлены.

## **4. Как вернуть несколько значений из функции Python?**

С помощью оператора return. Чтобы вернуть несколько значений, нужно написать их через запятую.

## **5. Какие существуют способы передачи значений в функцию?**

Существует два способа передачи параметров в функцию: по значению и по адресу. При передаче по значению на месте формальных параметров записываются имена фактических параметров. При вычислении функции в

стек заносятся копии значений фактических параметров, и операторы функции работают с этими копиями.

## **6. Как задать значение аргументов функции по умолчанию?**

В Python аргументам функции можно присваивать значения по умолчанию, используя оператор присваивания «=».

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

Лямбда-выражения на Python - конструкторы простых безымянных однострочных функций. Могут быть использованы везде, где требуется.

## **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). Этот PEP описывает соглашения, а не правила или синтаксис.

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

Одиночные строки документации предназначены для действительно очевидных случаев.

Многострочные строки документации состоят из однострочной строки документации с последующей пустой строкой, а затем более подробным описанием. Первая строка может быть использована автоматическими

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 открывающие кавычки, или на следующей строке. Вся документация должна иметь такой же отступ, как кавычки на первой строке.