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 №2.11
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: Замыкания в языке Python.

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

Ход работы:

Создание общедоступного репозитория на «GitHub», клонирование репозитория, редактирование файла «.gitignore», организация репозитория согласно модели ветвления «git-flow» (рис. 1).

```
C:\Program Files\Git>cd C:\Users\Andrey\Desktop\пайтон\14 лаба

C:\Users\Andrey\Desktop\пайтон\14 лаба>git clone https://github.com/AndreyPust/Python_laboratory_work_2.11.git
Cloning into 'Python_laboratory_work_2.11'...
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\14 лаба>cd C:\Users\Andrey\Desktop\пайтон\14 лаба\Python_laboratory_work_2.11

C:\Users\Andrey\Desktop\пайтон\14 лаба\Python_laboratory_work_2.11>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? [C:/Users/Andrey/Desktop/пайтон/14 лаба/Python_laboratory_work_2.11/.git/hooks]
```

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

Проработка примеров лабораторной работы:

Код программы (функций) показывающих применение разных переменных и примеров, представленных в лабораторной работе в том числе и функций, использующих замыкание и результаты работы программы (рис. 2, 3).

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


def add_two(a):
    """
    Функция позволяет складывать переданный ей аргумент с локальной
переменной x.
    Функция показывает пример локальной переменной (переменная x).
    """
```

```
    x = 2
    return a + x


def add_four(a):
    """
    Функция показывает пример использования области видимости enclosing.
    Переменная x имеет область видимости enclosing для функции add_some().
    """
    x = 2

    def add_some():
        print("x = " + str(x))
        return a + x
    return add_some()


example_x = 4  # глобальная переменная для функции fun()


def fun():
    """
    Функция показывает пример применения глобальной переменной.
    Глобальной переменной (Global) является example_x,
    так как она была объявлена в этом модуле (в файле с расширением .py).
    """
    print(example_x + 3)


def mul(a, b):
    """
    Функция перемножает переденные ей параметры a и b.
    """
    return a * b


def mul5(a):
    """
    Функция вызывает функцию mul() и передает ей параметр 5
    в добавление к переданному ей параметру a.
    """
    return mul(5, a)


def mul_5(a):
    """
    Функция с использованием замыкания.
    """
    def helper(b):
        return a * b
    return helper


def fun1(a):
    """
    Пример функции, использующей замыкание.
    В функции fun1() объявлена локальная переменная x, значение которой
определяется
    аргументом a. В функции fun2() используются эта же переменная x, nonlocal
указывает
    на то, что эта переменная не является локальной, следовательно, ее
значение будет
    взято из ближайшей области видимости, в которой существует переменная с
таким же именем.
```

```
    В нашем случае – это область enclosing, в которой этой переменной x
присваивается
    значение a * 3. Также как и в предыдущем случае, на переменную x после
вызова fun1(4),
    сохраняется ссылка, поэтому она не уничтожается.
    """
    x = a * 3

    def fun2(b):
        nonlocal x
        return b + x
    return fun2


if __name__ == "__main__":
    print(add_two(3))

    print(add_four(5))

    fun()

    print(mul(3, 4))

    print(mul5(7))

    new_mul5 = mul_5(5)
    new_mul5(2)
    new_mul5(7)

    test_fun = fun1(4)
    print(test_fun(7))
```

Рисунок 2 – Код программы примеров лабораторной работы.

```
C:\Users\Andrey\AppData\Local\Programs\Python\Python39\python.exe "C:\Users\A
5
x = 2
7
7
12
35
19

Process finished with exit code 0
```

Рисунок 3 – Результаты работы программы примеров.

Выполнение индивидуального задания:

Используя замыкания функций, объявите внутреннюю функцию, которая бы все повторяющиеся символы заменяла одним другим указанным символом. Какие повторяющиеся символы искать и на что заменять, определяются параметрами внешней функции. Внутренней функции

передается только строка для преобразования. Преобразованная (сформированная) строка должна возвращаться внутренней функцией. Вызовите внутреннюю функцию замыкания и отобразите на экране результат ее работы. (Вариант 26 (6)).

Код программы индивидуального задания и результаты работы программы (рис. 4, 5).

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


def replace_char(char_find, char_replace):
    """
    Функция производит замену последовательности искомых символов (char_find)
    на один указанный в переденных аргументах символ (char_replace).
    Для того, чтобы функции можно было передать новое предложение или строку
    при тех же искомых и меняемых символах, в функции используется замыкание,
    поэтому в дальнейшем, при вызове функции ей передается всего один
аргумент
    - новое предложение для замены.
    В качестве результата функция возвращает то, что возвращает вложанная
функция,
    а именно измененное предложение.
    """
    def in_func(proposal):
        result_proposal = ''
        prev_char = ''
        count = 0
        len_proposal = len(proposal)

        # Произведем замену последовательностей нужных символов.
        for char in proposal:
            if count == len_proposal - 1:
                break
            if char != char_find:
                result_proposal += char
            if char == char_find and char == prev_char and proposal[count +
1] != prev_char:
                result_proposal += char_replace

            prev_char = char
            count += 1
        return result_proposal

    return in_func


if __name__ == "__main__":
    # Используя замыкания функций, объявите внутреннюю функцию, которая бы
все
    # повторяющиеся символы заменяла одним другим указанным символом. Какие
    # повторяющиеся символы искать и на что заменять, определяются
параметрами
    # внешней функции. Внутренней функции передается только строка для
преобразования.
    # Преобразованная (сформированная) строка должна возвращаться внутренней
функцией.
```

```
    # Вызовите внутреннюю функцию замыкания и отобразите на экране результат
ее работы.
    # (Вариант 26 (6)).

    char_find_x = str(input("Введите символы, которые хотите заменить:"))
    char_replace_y = str(input("Введите символы, на которые необходимо
заменить "
                               "повторяющиеся искомые:"))

    replace_x_with_y = replace_char(char_find, char_replace)
    proposal_1 = str(input("Введите строку или предложение, в котором
необходимо "
                           "произвести замену:"))
    print("Итоговое предложение после замены:", replace_x_with_y(proposal_1))

    proposal_2 = str(input("Введите строку или предложение, в котором
необходимо "
                           "произвести замену:"))
    print("Итоговое предложение после замены:", replace_x_with_y(proposal_2))
```

Рисунок 10 – Код программы индивидуального задания.

```
C:\Users\Andrey\AppData\Local\Programs\Python\Python39\python.exe "C:\Users\Andrey\Desktop\най
Введите символы, которые хотите заменить:x
Введите символы, на которые необходимо заменить повторяющиеся искомые:y
Введите строку или предложение, в котором необходимо произвести замену:xxx000xxx000xxx
Итоговое предложение после замены: y000y000
Введите строку или предложение, в котором необходимо произвести замену:qwertyqwerty
Итоговое предложение после замены: qwertyqwert

Process finished with exit code 0
```

Рисунок 11 – Результаты работы программы индивидуального задания.

Ответы на контрольные вопросы:

1. Замыкание – функция, у которой есть доступ к области видимости, сформированной внешней по отношению к ней функции даже после того, как эта внешняя функция завершила работу. Это значит, что в замыкании могут храниться переменные, объявленные во внешней функции и переданные ей аргументы.

2. В Python вложенные функции имеют доступ к переменным внешней функции даже после того, как внешняя функция прекратила свою работу.

3. Область видимости Local имеют переменные, которые создаются и используются внутри функций, снаружи доступ к ним невозможен.

4. Внутри функции могут быть вложенные функции и локальные переменные. Эта локальная переменная функции для её вложенной функции находится в «enclosing» области видимости.

5. Переменные области видимости «global» – это глобальные переменные уровня модуля (файла с расширением .py).

6. Built-in – уровень Python интерпретатора, максимально широкая область видимости. В рамках этой области видимости находятся функции «open», «len» и т.п., также туда входят исключения. Эти сущности доступны в любом модуле Python и не требуют предварительного импорта.

7. Использовать замыкания необходимо для сохранения значений локальных переменных из внешней функции и использовать их во внутренней. Для этого необходимо вначале вызвать внешнюю функцию и передать ей аргументы и сохранить их в виде ссылки, а затем вызывать ссылку передавая ей аргументы для вложенной функции (в которой будут использоваться сохраненные таким образом и переменные «enclosing»).

8. В качестве примера построения иерархических данных можно привести код, в котором относительно операции объединения tpl, оказались замкнуты кортежи:

```
>>> tpl = lambda a, b: (a, b)
>>> b = tpl(3, a)
>>> b
(3, (1, 2))
>>> c = tpl(a, b)
>>> c
((1, 2), (3, (1, 2)))
```

Вывод: в ходе лабораторной работы были изучены замыкания и их применение в языке программирования Python. Также были более подробно изучены четыре области видимости переменных (local, global, enclosing, built-in). Были изучены способы создания замыканий в Python.