

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 №1**  
**дисциплины «Основы кроссплатформенного**  
**программирования»**

Выполнил:

Яхшибоев Элёр Содикжон угли

1 курс, группа ИТС-б-о-21-1,

11.03.02 «Инфокоммуникационные  
технологии и системы связи»,

направленность (профиль)

«Инфокоммуникационные системы и сети»,  
очная форма обучения

\_\_\_\_\_  
(подпись)

Руководитель практики:

Воронкин Р.А, канд. техн. наук, доцент  
кафедры инфокоммуникаций

\_\_\_\_\_  
(подпись)

Отчет защищен с оценкой \_\_\_\_\_ Дата защиты \_\_\_\_\_

Ставрополь, 2022 г.

## Тема: Замыкания в языке Python

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

### Краткий конспект

замыкание (*closure*) в программировании — это функция, в теле которой присутствуют ссылки на переменные, объявленные вне тела этой функции в окружающем коде и не являющиеся ее параметрами.

Обычно, по области видимости, переменные делят на глобальные и локальные. Глобальные существуют в течении всего времени выполнения программы, а локальные создаются внутри методов, функций и прочих блоках кода, при этом, после выхода из такого блока переменная удаляется из памяти.

### Область видимости *Local*

Эту область видимости имеют переменные, которые создаются и используются внутри функций.

#### Пример

```
>>> def add_two(a):
```

```
    x = 2
```

```
    return a + x
```

```
>>> add_two(3)
```

```
5 >
```

```
>> print(x)
```

```
Traceback (most recent call last):
```

```
File "<pyshell#5>", line 1, in <module>
```

```
    print(x)
```

```
NameError: name 'x' is not defined
```

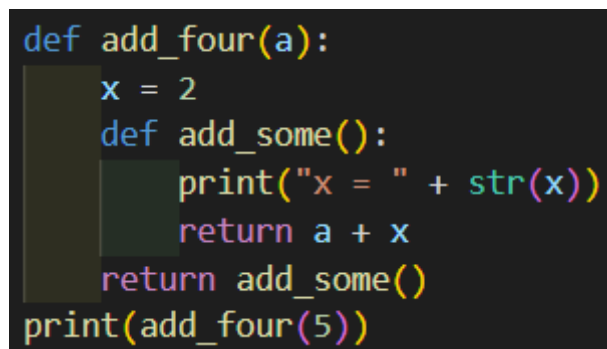
В данной программе объявлена функция *add\_two()*, которая прибавляет двойку к переданному ей числу и возвращает полученный результат. Внутри этой функции используется переменная *x*, доступ к которой снаружи

невозможен. К тому же, эта переменная удаляется из памяти каждый раз (во всяком случае, должна удаляться), когда завершается *add\_two()*.

**Область видимости *Enclosing*** Суть данной области видимости в том, что внутри функции могут быть вложенные функции и локальные переменные, так вот локальная переменная функции для ее вложенной функции находится в *enclosing* области видимости.

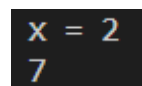
Пример.

```
>>> def add_four(a):  
    x = 2  
    def add_some():  
        print("x = " + str(x))  
        return a + x  
    return add_some()  
  
>>> add_four(5)  
x = 2  
7
```



```
def add_four(a):  
    x = 2  
    def add_some():  
        print("x = " + str(x))  
        return a + x  
    return add_some()  
print(add_four(5))
```

Рис 1. Код программы



```
x = 2  
7
```

Рис 2. Ее результат

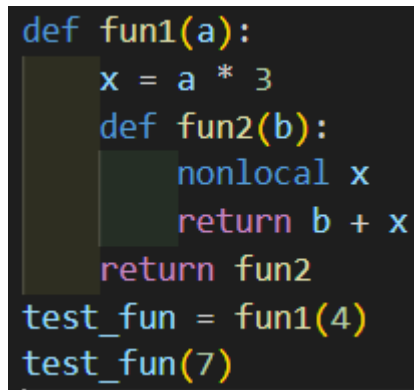
В данном случае переменная *x* имеет область видимости *enclosing* для функции *add\_some()*.

```
>>> def fun1(a):  
    x = a * 3  
    def fun2(b):
```

```

        nonlocal x
    return b + x
return fun2
>>> test_fun = fun1(4)
>>> test_fun(7)
19

```



```

def fun1(a):
    x = a * 3
    def fun2(b):
        nonlocal x
        return b + x
    return fun2
test_fun = fun1(4)
test_fun(7)

```

Рис 1. Код программы

19

Рис 4. Ее результат

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

### Область видимости *Global*

Переменные области видимости *global* – это глобальные переменные уровня модуля (модуль – это файл с расширением *.py*).

Пример.

```

>>> x = 4
>>> def fun():
print(x+3)

```

```
>>> fun()
```

```
7
```

В приведенном выше коде переменная *x* – это *global* переменная. Доступ к ней можно получить из любой функции, объявленной в данном модуле. Но если мы этот модуль импортируем в каком, то другом модуле, то *x* для него уже не будет переменной уровня *global*.

### **Область видимости *Built-in***

Уровень *Python* интерпретатора. 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. *Built-in* – это максимально широкая область видимости.

### **Ход работы:**

#### **Задание 6**

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.

```

1  # !/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  import sys
5
6  def fun1(to_replace, replacer):
7
8      def fun2(string):
9          nonlocal to_replace, replacer
10         result = string.replace(replacer, to_replace)
11         return result
12
13     return fun2
14
15
16 if __name__ == "__main__":
17     x = input("Введите строку: ")
18     c = input("Введите символ, который нужно заменить: ")
19     h = input("Введите символ, на который заменить: ")
20     rep = fun1(h, c)
21     print(rep(x))

```

Рис 1. Код программы

```

Введите строку: 1234*6789
Введите символ, который нужно заменить: *
Введите символ, на который заменить: 5
123456789

```

Рис 2. Результат программы

### Вопросы:

1. Что такое замыкание?

Замыкание (closure) в программировании – это функция, в теле которого присутствуют ссылки на переменные, объявленные вне тела этой функции в окружающем коде и не являющейся ее параметрами. Обычно, по области видимости, переменные делят на глобальные и локальные.

2. Как реализованы замыкания в языке программирования Python?

```

def outer():
    x = 1
    def inner():
        print(f'x in outer function: {x}')
    return inner

```

Функция `outer` определяется с функцией `inner` внутри, а функция `outer` возвращает функцию `inner`; именно она – возвращаемое значение `outer`. Здесь вложенная функция – это и есть замыкание.

3. Что подразумевает под собой область видимости *Local*?

Область видимости *Local* имеют переменные, которые создаются и используются внутри функции.

4. Что подразумевает под собой область видимости *Enclosing*?

Суть области видимости *Enclosing* в том, что внутри функции могут быть вложенные функции и локальные переменные, так вот локальная

5. Что подразумевает под собой область видимости *Global*?

Область видимости *Global* Переменные области видимости *global* – это глобальные переменные уровня модуля (модуль – это файл с расширением *.py*).

6. Что подразумевает под собой область видимости *Built-in*?

**Область видимости *Built-in*** Уровень *Python* интерпретатора. 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. *Built-in* – это максимально широкая область видимости.

7. Как использовать замыкания в языке программирования *Python*?

```
>>> new_mul5 = mul(5)
>>> new_mul5
<function mul.<locals>.helper at 0x000001A7548C1158>
>>> new_mul5(2)
10
>>> new_mul5(7)
35
```

Вызывая *new\_mul5(2)*, мы фактически обращаемся к функции *helper()*, которая находится внутри *mul()*. Переменная *a*, является локальной для *mul()*, и имеет область *enclosing* в *helper()*. Несмотря на то, что *mul()* завершила свою

работу, переменная а не уничтожается, т.к. на нее сохраняется ссылка во внутренней функции, которая была возвращена в качестве результата.

8. Как замыкания могут быть использованы для построения иерархических данных?

Перейдем с уровня математики на уровень функционального программирования. Вот как определяется “свойство замыкания” в книге “Структура и интерпретация компьютерных программ” Айбельсона Х., Сассмана Д. Д.: “В общем случае, операция комбинирования объектов данных обладает свойством замыкания в том случае, если результаты соединения объектов с помощью этой операции сами могут соединяться этой же операцией”. Это свойство позволяет строить иерархические структуры данных.

**Вывод:** в ходе выполнения лабораторной работы было изучены Замыкания в пайтоне, а также мы приобрели навыки по работе с замыканиями при написании программ с помощью пайтон.