Министерство науки и высшего образования Российской Федерации Федеральное государственное автономное образовательное учреждение высшего образования

«СЕВЕРО-КАВКАЗСКИЙФЕДЕРАЛЬНЫЙУНИВЕРСИТЕТ»

Институт цифрового развития

Кафедра инфо коммуникаций

# ОТЧЕТ

**ПО ЛАБОРАТОРНОЙ РАБОТЕ№2.11**

# Дисциплины « Основы кроссплатформенного программирования

|  |
| --- |
| Выполнил:  Гуляницкий Александр Евгеньевич  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 ка федры инфокоммуникаций  (подпись) |

Отчет защищен с оценкой Дата защиты

Ставрополь 2022 г

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

# Ход работы:

Создал новый репозиторий <https://github.com/Alexander-its/laba-2.11>

и начал работать с примерами.

Пример 1

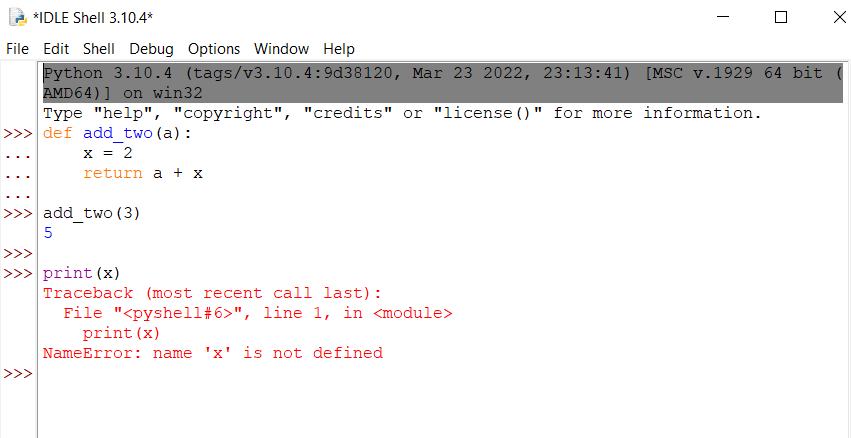
****

Рис 1 работа с Python

Пример 2

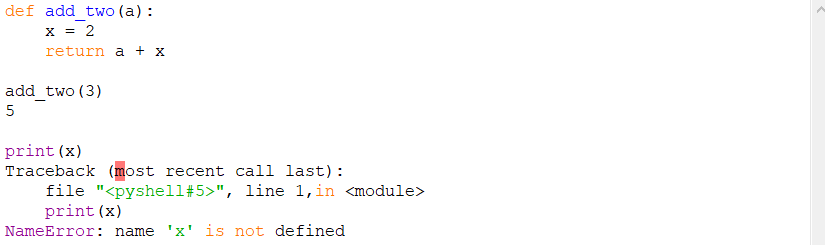


Рис 2 работа с Python

# Индивидуальное задание

# 8 Вариант.

# Используя замыкания функций, объявите внутреннюю функцию, которая принимает два параметра a , b , а затем, возвращает строку в формате: «Для значений a, b функция f(a,b) = » где число – это вычисленное значение функции f Ссылка на f передается как аргумент внешней функции. Вызовите внутреннюю функцию замыкания и отобразите на экране результат ее работы. Функцию f придумайте самостоятельно (она должна что то делать с двумя параметрами a , b и возвращать результат).

# C:\Users\GO_PB\Pictures\Screenshots\Снимок экрана (401).png

# C:\Users\GO_PB\Pictures\Screenshots\Снимок экрана (402).png

# Рис 3 Результат индивидуального задания.

**Вывод:** Я приобрёл навыки по работе с замыканиями при написании программ с помощью языка программирования Python.

# Контрольные вопросы:

1. Что такое замыкание?

Замыкания — это функции, ссылающиеся на независимые (свободные) переменные. Другими словами, функция, определённая в замыкании, «запоминает» окружение, в котором она была создана.

1. Как реализованы замыкания в языке программирования Python?

Замыкание (closure) — функция, которая находится внутри другой функции и ссылается на переменные объявленные в теле внешней функции (свободные переменные). Внутренняя функция создается каждый раз во время выполнения внешней

1. Что подразумевает под собой область видимости Local?

Локальная **область видимости** наиболее часто используется в Python. Когда мы создаем переменную в блоке кода, она будет разрешена при помощи ближайшей области видимости, или областей. Группирование всех этих областей известно как среда блоков кода. Другими словами, все назначения выполняются в **локальной области** по умолчанию.

1. Что подразумевает под собой 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.

5. Что подразумевает под собой область видимости Global?

Python содержит **оператор global**. Это ключевое слово Python. Оператор **global** объявляет переменную доступной для блока кода, следующим за оператором. Хотя вы и можете создать наименование, перед тем, как объявить его глобальным, я настоятельно не рекомендую этого делать.

1. Что подразумевает под собой область видимости Build-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

1. Как использовать замыкания в языке программирования Python?

Для начала разберем следующий пример.

**>>> def mul(a, b):**

**return a \* b**

**>>> mul(3, 4)**

**12**

Функция *mul()* умножает два числа и возвращает полученный результат. Если мы ходим на базе нее решить задачу: “умножить число на пять”, то в самом простом случае, можно вызывать *mul()*, передавая в качестве первого аргумента пятерку.

**>>> mul(5, 2)**

**>>> mul(5, 7)**

**35**

Это неудобно. На самом деле мы можем создать новую функцию, которая будет вызывать *mul()*, с пятеркой и ещё одним числом, которое она будет получать в качестве своего единственного аргумента.

**>>> def mul5(a):**

**return mul(5, a)**

**>>> mul5(2)**

**10**

**>>> mul5(7)**

**35**

Уже лучше, но все равно пока не достаточно гибко, т.к. в следующий раз, когда нужно будет построить умножитель на семь, нам придется создавать новую функцию. Для решения этой проблемы воспользуемся замыканием.

**>>> def mul(a):**

**def helper(b):**

**return a \* b**

**return helper**

Вычислим выражение “5 \* 2 = ?” с помощью этой функции.

**>>> mul(5)(2)**

**10**

Создадим функцию – аналог *mul5()*.

**>>> new\_mul5 = mul(5)**

**>>> new\_mul5**

**<function mul.<locals>.helper at 0x000001A7548C1158>**

**>>> new\_mul5(2)**

**>>> 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 ***a***не уничтожается, т.к. на нее сохраняется ссылка во внутренней функции, которая была возвращена в качестве результата.

Рассмотрим ещё один пример.

**>>> def fun1(a):**

**x = a \* 3**

**def fun2(b):**

**nonlocal x**

**return b + x**

**return fun2**

**>>> test\_fun = fun1(4)**

**>>> test\_fun(7)**

**19**

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.

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

Сразу хочу сказать, что “свойство замыкания” – это не то замыкание, которое мы разобрали выше. Начнем разбор данного термина с математической точки зрения, а точнее с алгебраической. Предметом алгебры является изучение алгебраических структур – множеств с определенными на них операциями. Под множеством обычно понимается совокупность определенных объектов. Наиболее простым примером числового множества, является множество натуральных чисел. Оно содержит следующие числа: 1, 2, 3, … и т.д. до бесконечности. Иногда, к этому множеству относят число ноль, но мы не будем этого делать. Над элементами этого множества можно производить различные операции, например сложение:

**1 + 2 = 3**

Какие бы натуральные числа мы не складывали, всегда будем получать натуральное число. С умножением точно также. Но с вычитанием и делением это условие не выполняется.

**2 – 5 = -3**

Среди натуральных чисел нет числа -3, для того, чтобы можно было использовать вычитание без ограничений, нам необходимо расширить множество натуральных чисел до множества целых чисел:

**-∞, …, -2, -1, 0, 1, 2, …, ∞.**

Таким образом, можно сказать, что множество натуральных чисел замкнуто относительно операции сложения – какие бы натуральные числа мы не складывали, получим натуральное число, но это множество не замкнуто относительно операции вычитания.

Теперь 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. Покажем это на примере кортежей в *Python*.

Создадим функцию tpl(), которая на вход принимает два аргумента и возвращает кортеж. Эта функция реализует операцию “объединения элементов в кортеж”.

**>>> tpl = lambda a, b: (a, b)**

Если мы передадим в качестве аргументов числа, то, получим простой кортеж.

**>>> a = tpl(1, 2)**

**>>> a**

**(1, 2)**

Эту операцию можно производить не только над числами, но и над сущностями, ей же и порожденными.

**>>> b = tpl(3, a)**

**>>> b**

**(3, (1, 2))**

**>>> c = tpl(a, b)**

**>>> c**

**((1, 2), (3, (1, 2)))**

Таким образом, в нашем примере кортежи оказались замкнуты относительно операции объединения tpl. Вспомните аналогию с натуральными числами, замкнутыми относительно сложения.