

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙ-
СКОЙ ФЕДЕРАЦИИ**

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

Кафедра инфокоммуникаций

Основы кроссплатформенного программирования

Отчет по лабораторной работе №2.11

Тема: «Замыкания в языке Python»

Выполнил студент группы

ИВТ-б-о-21-1

Криворот В.Г. « » _____ 20__ г.

Подпись студента _____

Работа защищена « » _____ 20__ г.

Проверил доцент

Кафедры инфокоммуникаций, старший
преподаватель

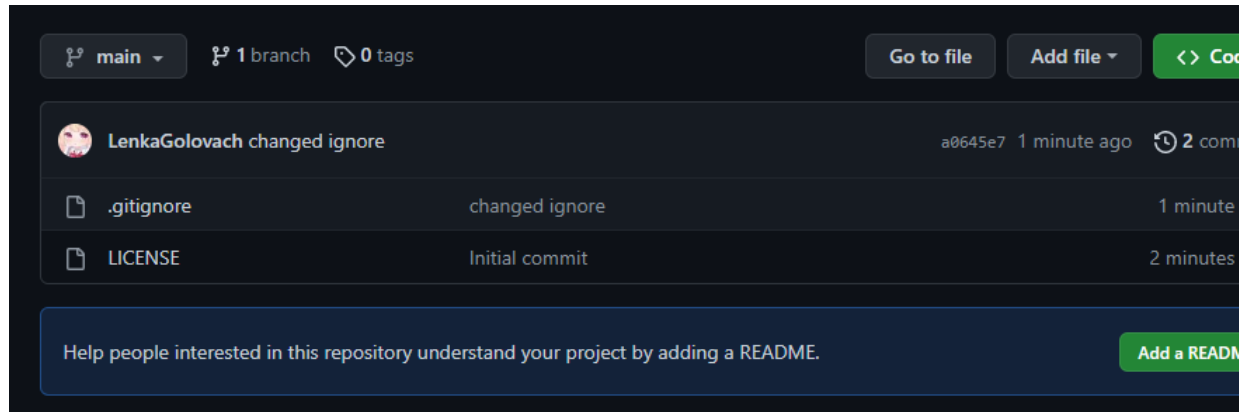
Воронкин Р.А.

(подпись)

Ставрополь 2022

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

Ход работы:



1.

Создал репозиторий в GitHub, дополнил правила в `.gitignore` для ра-боты с IDE PyCharm с ЯП Python, выбрал лицензию MIT, клонировал его на компьютер и организовал в соответствии с моделью ветвления git-flow.

Рисунок 1.1 – Созданный репозиторий



Рисунок 1.2 – Дополнил правила в `.gitignore`

```

c:\Users\Admin\Desktop\git\Python6>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/Admin/Desktop/git/Python6/.git/hooks]

c:\Users\Admin\Desktop\git\Python6>

```

Рисунок 1.3 – Организация репозитория в соответствии с моделью ветвления git-flow

2. Создал проект Pycharm в папке репозитория, проработал примеры ЛР.

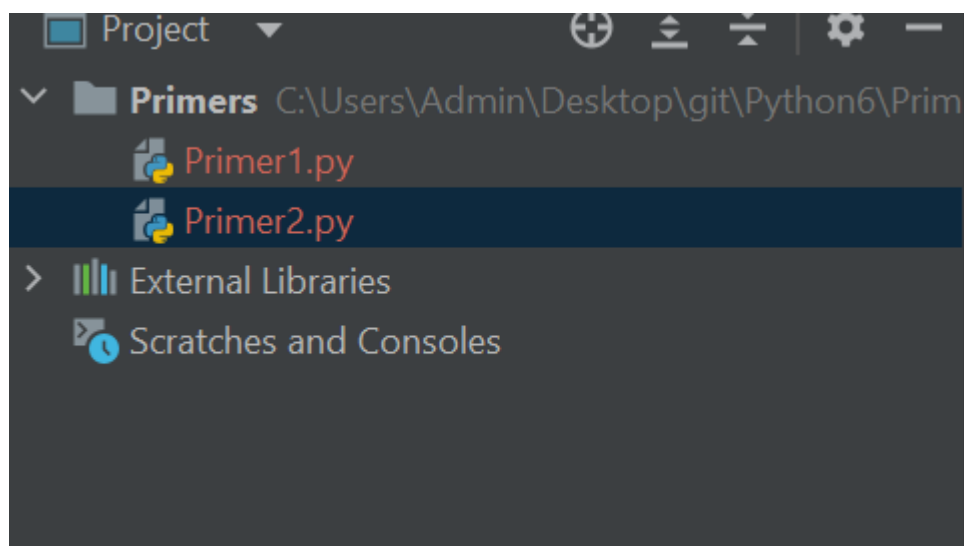


Рисунок 2.1 – Созданные проекты

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

def mul(a):
    def helper(b):
        return a * b
    return helper

if __name__ == '__main__':
    print(mul(5)(2))
```

Primer1 x

C:\Users\Admin\AppData\Local\Programs\Python

10

Process finished with exit code 0

Рисунок 2.2 – Результат выполнения примера №1

```
Primer1.py x Primer2.py x
def fun1(a):
    x = a * 3

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

if __name__ == "__main__":
    test_fun = fun1(4)
    print(test_fun(7))

if __name__ == "__main__"
```

Primer2 x

C:\Users\Admin\AppData\Local\Progra

19

Process finished with exit code 0

Рисунок 2.3 – Результат выполнения примера №2

3. Индивидуально задание. Вариант – 16(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.

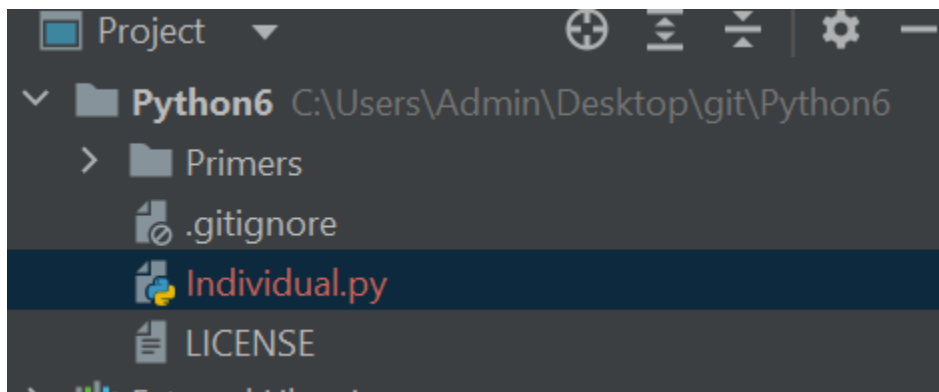


Рисунок 3.1 – Созданный проект

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Используя замыкания функций, определите вложенную функцию, которая бы
увеличивала значение переданного параметра на 3 и возвращала бы вычисленный результат.
Вызовите внешнюю функцию для получения ссылки на внутреннюю функцию и присвойте ее переменной с именем cnt.
Затем, вызовите внутреннюю функцию через переменную cnt со значением k, введенным с клавиатуры.
"""

def fun1(to_replace, replacer):
    def fun2(string):
        nonlocal to_replace, replacer
        result = string.replace(replacer, to_replace)
        return result

    return fun2

if __name__ == "__main__":
    a = input("Строка: ")
    b = input("Введите символ, который нужно заменить: ")
    c = input("Введите символ, на который заменить: ")
    rep = fun1(c, b)
    print(rep(a))
```

Рисунок 3.2 – Результат выполнения программы

Вывод: в результате выполнения лабораторной работы были приобре-

тены практические навыки и теоретические сведения по работе с замыканиями

при написании программ с помощью языка программирования Python версии 3.x..

Ответы на контрольные вопросы:

1. Что такое замыкание?

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.

2. Как реализованы замыкания в языке программирования Python?

Замыкания в Python реализованы посредством манипулирования областью видимости функций.

3. Что подразумевает под собой область видимости Local?

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

```
>>> 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
```

Пример. 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()`.

4. 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.

```
>>> 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
```

5. Что подразумевает под собой область видимости Global?

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

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


6. 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.

7. Как использовать замыкания в языке программирования Python?

Для создания замыкания в Python, должны быть выполнены следующие пункты:

- У нас должна быть вложенная функция (функция внутри функции);
- вложенная функция должна ссылаться на значение, определенное в объемлющей функции;
- объемлющая функция должна возвращать вложенную функцию.

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

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.

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.

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:

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”**.