

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ  
Федеральное государственное автономное  
образовательное учреждение высшего образования  
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

**Кафедра инфокоммуникаций**

**Отчет по лабораторной работе № 2.11  
по дисциплине «Основы программной инженерии»**

Выполнил студент группы ПИЖ-б-о-22-1

Душин Александр Владимирович.

Подпись студента \_\_\_\_\_

Работа защищена « » \_\_\_\_\_ 20\_\_ г.

Проверил Воронкин Р.А. \_\_\_\_\_  
(подпись)

Ставрополь 2023

Тема: Замыкания в языке Python.

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

Ход выполнения работы:

1. Создать общедоступный репозиторий на GitHub с использованием лицензии MIT и язык программирования Python:


## Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere?

[Import a repository.](#)

Required fields are marked with an asterisk (\*).

Owner \*

 MrPlatynum ▾

Repository name \*

ProgrammEngineering14

✔ ProgrammEngineering14 is available.

Great repository names are short and memorable. Need inspiration? How about

[literate-octo-computing-machine](#) ?

Description (optional)



Public

Anyone on the internet can see this repository. You choose who can commit.



Private

You choose who can see and commit to this repository.

Initialize this repository with:



Add a README file

This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore

.gitignore template: Python ▾

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license

License: MIT License ▾

A license tells others what they can and can't do with your code. [Learn more about licenses.](#)



You are creating a public repository in your personal account.

Create repository

Рисунок 1 – Создание общедоступного репозитория на GitHub с заданными настройками

```
Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents
$ git clone https://github.com/MrPlatynum/ProgrammEngineering14.git
Cloning into 'ProgrammEngineering14'...
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (4/4), done.
```

Рисунок 2 – Клонирование созданного репозитория на локальный компьютер

```
env.bak/
venv.bak/

# Spyder project settings
.spyderproject
.spyproject

# Rope project settings
.ropeproject

# makedocs documentation
/site

# mypy
.mypy_cache/
.dmypy.json
dmypy.json

# Pyre type checker
.pyre/

# pytype static type analyzer
.pytype/

# Cython debug symbols
cython_debug/

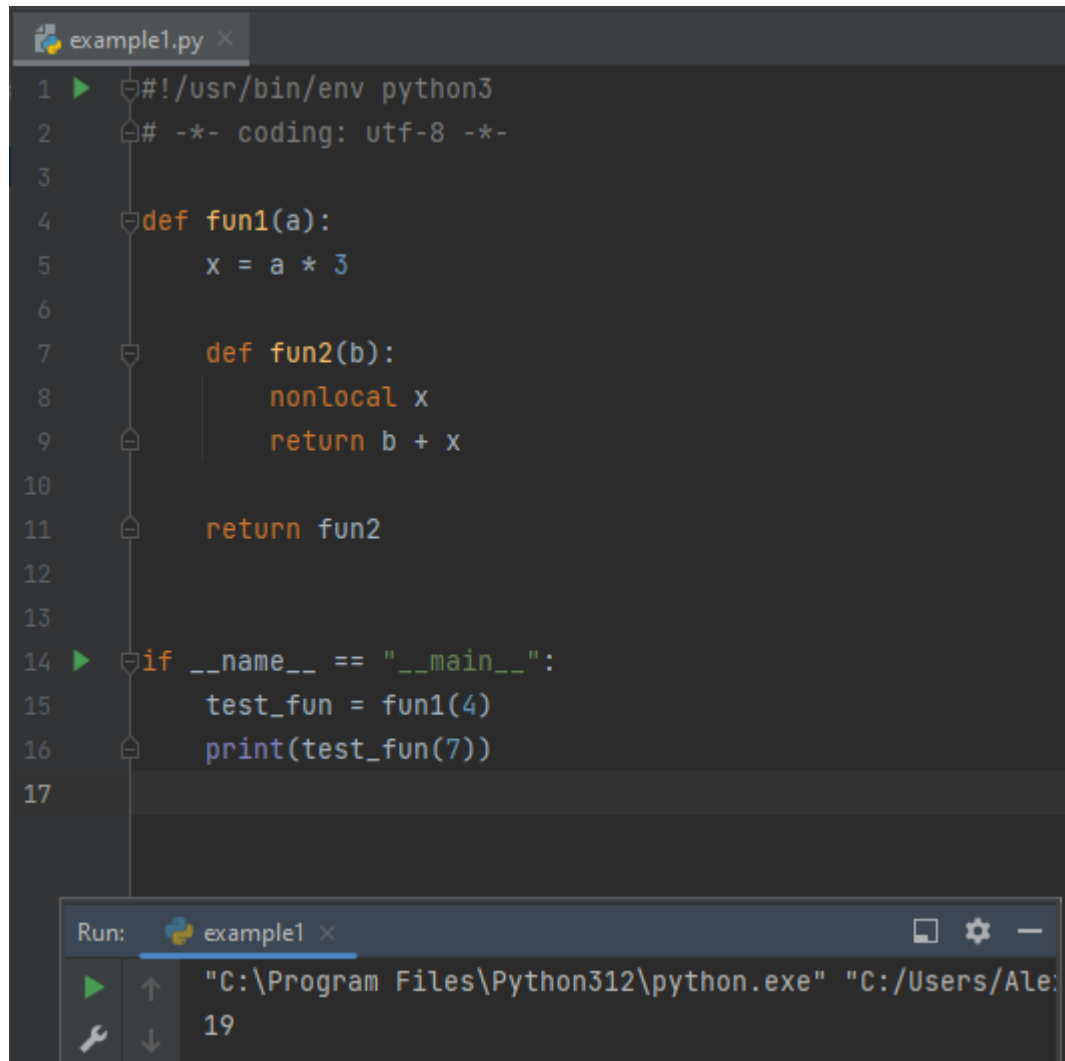
# PyCharm
# JetBrains specific template is maintained in a separate JetBrains.gitignore that can
# be found at https://github.com/github/gitignore/blob/main/Global/JetBrains.gitignore
# and can be added to the global gitignore or merged into this file. For a more nuclear
# option (not recommended) you can uncomment the following to ignore the entire idea folder.
.idea/
```

Рисунок 3 – файл .gitignore

```
Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents/ProgrammEngineering14 (main)
$ git checkout -b develop
Switched to a new branch 'develop'
```

Рисунок 4 – организация репозитория в соответствии с моделью ветвления git flow

2. Проработать примеры лабораторной работы, оформляя код согласно РЕР-8:



```
example1.py x
1  ▶  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  def fun1(a):
5      x = a * 3
6
7      def fun2(b):
8          nonlocal x
9          return b + x
10
11     return fun2
12
13
14  ▶  if __name__ == "__main__":
15      test_fun = fun1(4)
16      print(test_fun(7))
17
Run: example1 x
▶  "C:\Program Files\Python312\python.exe" "C:/Users/Ale:
19
```

Рисунок 5 – Пример №1

3. 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:

```

individual1.py x
2  # -*- coding: utf-8 -*-
3
4
5  def replace_repeated_chars(replace_char):
6      def inner_function(input_string):
7          result = ''
8          prev_char = ''
9
10         for char in input_string:
11             if char != prev_char:
12                 result += char
13                 prev_char = char
14             else:
15                 result += replace_char
16
17         return result
18
19     return inner_function
20
21
22  if __name__ == "__main__":
23      replace_func = replace_repeated_chars(
24          input('Введите символ, на который необходимо заменить повторяющиеся символы: '))
25      result = replace_func(input('Введите строку: '))
26      print(result) # выводим результат работы внутренней функции
27

```

Рисунок 6 – Решение индивидуального задания

```

"C:\Program Files\Python312\python.exe" "C:/Users/Alexander/Desktop/Универ/3
Введите символ, на который необходимо заменить повторяющиеся символы: *
Введите строку: Heeellloo Worldd
He**l**o** World*

```

Рисунок 7 – Вывод программы

4. Зафиксируем сделанные изменения, сольем ветки и отправим на удаленный репозиторий:

```

Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents/ProgrammEngineering14 (develop)
$ git log --oneline
677845d (HEAD -> develop) Финальные изменения
a82ab39 (origin/main, origin/HEAD, main) Initial commit

```

Рисунок 8 – Коммиты ветки develop во время выполнения лабораторной работы

```

Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents/ProgrammEngineering14 (develop)
$ git checkout main
Switched to branch 'main'
M       .gitignore
Your branch is up to date with 'origin/main'.

Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents/ProgrammEngineering14 (main)
$ git merge develop
Updating a82ab39..677845d
Fast-forward
 example1.py    | 16 ++++++
 individual1.py | 26 ++++++
 2 files changed, 42 insertions(+)
 create mode 100644 example1.py
 create mode 100644 individual1.py

Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents/ProgrammEngineering14 (main)
$ |

```

Рисунок 9 – Слияние ветки develop в ветку main

```

Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents/ProgrammEngineering14 (main)
$ git push origin main
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 12 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 942 bytes | 942.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/MrPlatynum/ProgrammEngineering14.git
 a82ab39..677845d  main -> main

```

Рисунок 10 – Отправка на удаленный репозиторий

Ответы на контрольные вопросы:

1. Что такое замыкание?

Замыкание (closure) – это функция, которая запоминает окружение, в котором она была создана. Она имеет доступ к переменным из этого окружения даже после того, как внешняя функция завершила свою работу.

2. Как реализованы замыкания в языке программирования Python?

В Python замыкания реализуются путем вложения функций: функция объявляется внутри другой функции и использует переменные из внешней функции. После выполнения внешней функции внутренняя функция все еще может использовать эти переменные.

3. Что подразумевает под собой область видимости Local?

Локальная область видимости охватывает переменные, определенные внутри функции и доступные только внутри этой функции.

4. Что подразумевает под собой область видимости Enclosing?

Область охвата (enclosing) описывает область видимости переменных во внешних функциях, к которым есть доступ из внутренней функции. Это позволяет внутренней функции использовать переменные из внешней функции, в которой она была определена.

5. Что подразумевает под собой область видимости Global?

Глобальная область видимости охватывает переменные, объявленные в основном теле программы или модуля. Эти переменные доступны из любой функции или блока кода в этом модуле.

6. Что подразумевает под собой область видимости Build-in?

Встроенная область видимости в Python содержит встроенные функции, имена которых предопределены в языке (например, `print()`, `len()`, `str()`). Они доступны в любом месте кода без необходимости импортирования.

7. Как использовать замыкания в языке программирования Python?

Замыкания в Python создаются путем вложения функций, когда внутренняя функция использует переменные из внешней функции. Пример использования замыкания был приведен в предыдущих ответах.

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

Замыкания могут использоваться для построения иерархических структур данных, например, для создания вложенных функций или объектов. Они могут хранить состояние или конфигурацию внешней функции и применять это состояние к внутренним функциям, обеспечивая уникальные контексты и поведение. Например, можно использовать замыкания для создания деревьев или вложенных структур данных, где внутренние функции или объекты содержат информацию о своем родителе или предыдущих состояниях.