

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 №2.11
дисциплины «Основы программной инженерии»

Выполнил:
Юрьев Илья Евгеньевич
2 курс, группа ПИЖ-б-о-22-1,
09.03.04 «Программная инженерия»,
направленность (профиль) «Разработка
и сопровождение программного
обеспечения», очная форма обучения

(подпись)

Руководитель практики:
Богданов С.С., ассистент кафедры
инфокоммуникаций

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

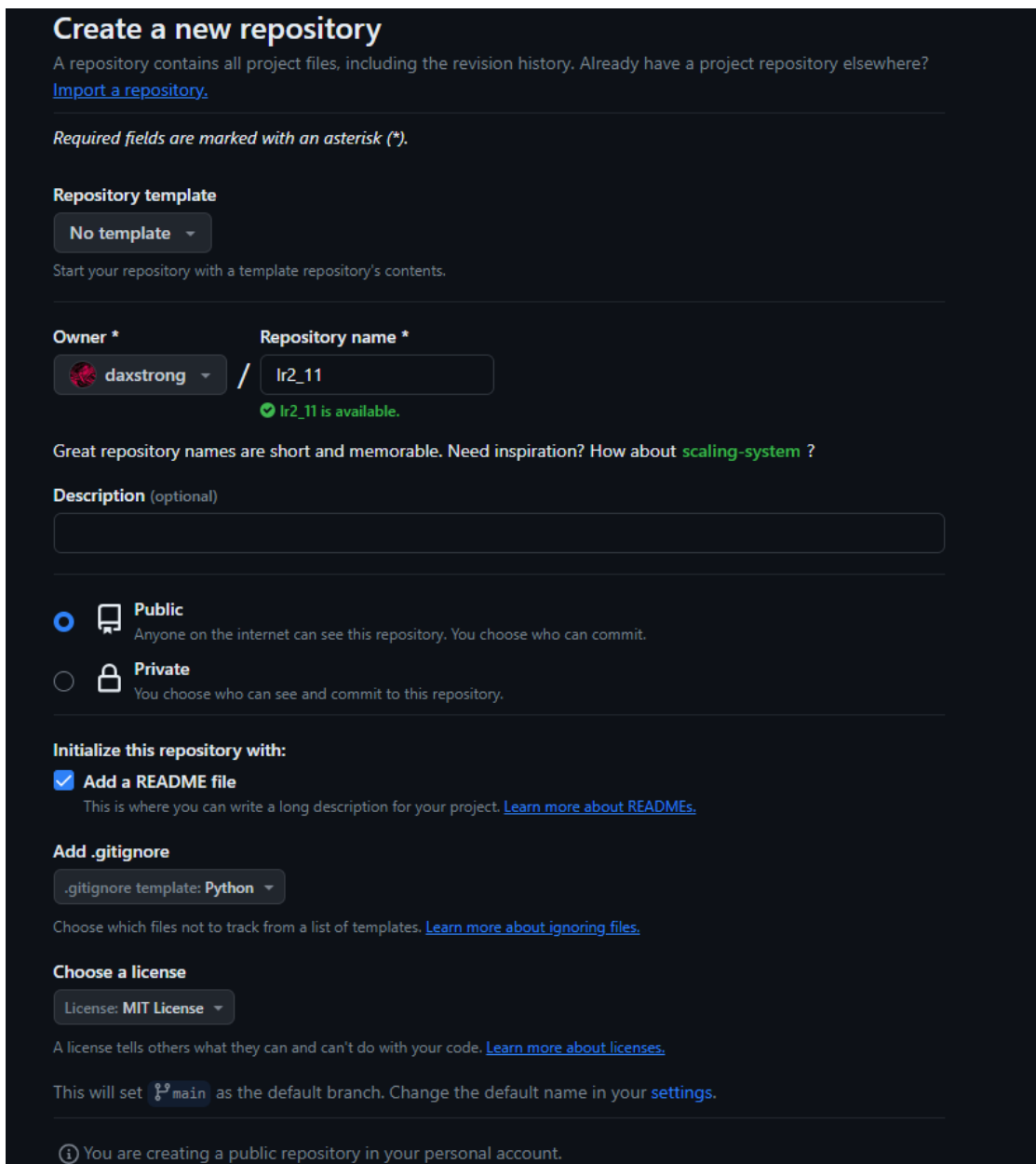
Ставрополь, 2023 г.

Тема: Замыкания в языке Python.

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

Ход выполнения работы:

1. Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия 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 ().*

Repository template

No template ▾

Start your repository with a template repository's contents.

Owner * daxstrong ▾ / **Repository name *** lr2_11

✔ lr2_11 is available.

Great repository names are short and memorable. Need inspiration? How about [scaling-system](#) ?

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.](#)

This will set `main` as the default branch. Change the default name in your [settings](#).

① You are creating a public repository in your personal account.

Рисунок 1 – Создание репозитория с заданными настройками

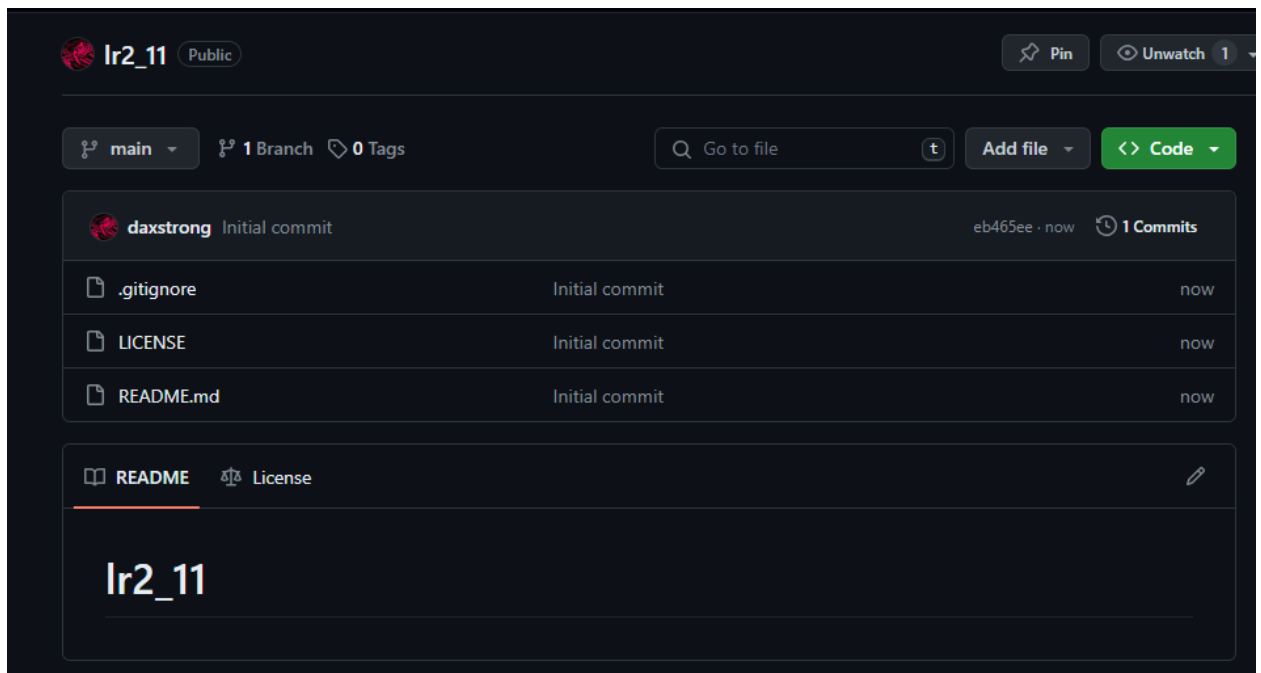


Рисунок 2 – Созданный репозиторий

```
ilyay@DESKTOP-FF1JT6S MINGW64 ~/OneDrive/Рабочий стол/Основы программной инженерии
$ git clone https://github.com/daxstrong/lr2_11.git
Cloning into 'lr2_11'...
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (5/5), done.
```

Рисунок 3 – Клонирование репозитория

```
ilyay@DESKTOP-FF1JT6S MINGW64 ~/OneDrive/Рабочий стол/Основы программной инженерии/lr2_11 (main)
$ git checkout -b develop
Switched to a new branch 'develop'
```

Рисунок 4 – Создание ветки develop

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

```

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))

```

Рисунок 5 – Пример №1

```

C:\Users\ilyay\AppData\Local\Microsoft\WindowsApps\python3.11.exe
19

```

Рисунок 6 – Вывод программы (Пример №1)

3. Выполним индивидуальное задание:

Используя замыкания функций, объявите внутреннюю функцию, которая заключает строку *s* (*s* – строка, параметр внутренней функции) в произвольный тег, содержащийся в переменной *tag* – параметре внешней функции. Далее, на вход программы поступает две строки: первая с тегом, вторая с некоторым содержимым. Вторую строку нужно поместить в тег из первой строки с помощью реализованного замыкания. Результат выведите на экран.

```

1 ▶ 1 #!/usr/bin/env python3
2   2 # -*- coding: utf-8 -*-
3
4   3 def wrap_in_tag(tag):
5       4 def wrap_string(s):
6           5 return f"<{tag}>{s}</{tag}>"
7
8       6 return wrap_string
9
10
11 ▶ 11 if __name__ == "__main__":
12     12 # Ввод данных
13     13 tag_input = input("Введите тег: ")
14     14 content_input = input("Введите содержимое: ")
15
16     15 # Создание и использование замыкания
17     16 result_closure = wrap_in_tag(tag_input)
18     17 result = result_closure(content_input)
19
20     18 # Вывод результата
21     19 print("Результат:", result)

```

Рисунок 7 – Решение индивидуального задания

```

C:\Users\ilyay\AppData\Local\Microsoft\WindowsApps\python3.11.exe
Введите тег: em
Введите содержимое: Python
Результат: <em>Python</em>

```

Рисунок 8 – Вывод программы (Индивидуальное задание)

Объяснение кода:

1. `wrap_in_tag` - это внешняя функция, которая принимает тег и возвращает внутреннюю функцию `wrap_string`.
2. `wrap_string` - это внутренняя функция, которая принимает строку `s` и использует тег из внешней функции, чтобы обернуть эту строку в тег. Значение `tag` сохранено из внешней функции `wrap_in_tag`.

3. Создание замыкания: когда мы вызываем `wrap_in_tag(tag_input)`, мы создаем экземпляр внутренней функции `wrap_string`, который "замыкает" значение `tag_input` внутри себя.

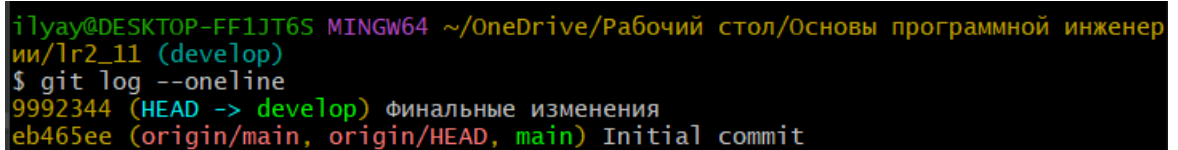
4. `result_closure` теперь представляет собой функцию, которая ожидает строку и будет использовать тег, сохраненный при создании замыкания.

5. Использование замыкания: при вызове `result_closure(content_input)`, замыкание использует сохраненное значение `tag_input` для обертывания введенного содержимого в тег.

6. Результат сохраняется в переменной `result` и выводится на экран.

Таким образом, замыкание позволяет внутренней функции `wrap_string` использовать значение `tag`, сохраненное из внешней функции `wrap_in_tag`.`

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



```
ilyay@DESKTOP-FF1JT6S MINGW64 ~/OneDrive/Рабочий стол/Основы программной инженерии/lr2_11 (develop)
$ git log --oneline
9992344 (HEAD -> develop) финальные изменения
eb465ee (origin/main, origin/HEAD, main) Initial commit
```

Рисунок 15 – Коммиты ветки `develop` во время выполнения лабораторной Работы

```

ilyay@DESKTOP-FF1JT6S MINGW64 ~/OneDrive/Рабочий стол/Основы программной инженерии/lr2_11 (develop)
$ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.

ilyay@DESKTOP-FF1JT6S MINGW64 ~/OneDrive/Рабочий стол/Основы программной инженерии/lr2_11 (main)
$ git merge develop
Updating eb465ee..9992344
Fast-forward
 .idea/.gitignore           | 8 ++++++++
 .idea/inspectionProfiles/profiles_settings.xml | 6 +++++
 .idea/lr2_11.iml          | 8 ++++++++
 .idea/misc.xml             | 4 ++++
 .idea/modules.xml         | 8 ++++++++
 .idea/vcs.xml              | 6 +++++
 ex1.py                     | 16 ++++++
 individual.py              | 21 ++++++
8 files changed, 77 insertions(+)
create mode 100644 .idea/.gitignore
create mode 100644 .idea/inspectionProfiles/profiles_settings.xml
create mode 100644 .idea/lr2_11.iml
create mode 100644 .idea/misc.xml
create mode 100644 .idea/modules.xml
create mode 100644 .idea/vcs.xml
create mode 100644 ex1.py
create mode 100644 individual.py

```

Рисунок 16 – Слияние веток main и develop

```

ilyay@DESKTOP-FF1JT6S MINGW64 ~/OneDrive/Рабочий стол/Основы программной инженерии/lr2_11 (main)
$ git push origin main
Enumerating objects: 13, done.
Counting objects: 100% (13/13), done.
Delta compression using up to 12 threads
Compressing objects: 100% (11/11), done.
Writing objects: 100% (12/12), 2.09 KiB | 2.09 MiB/s, done.
Total 12 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/daxstrong/lr2_11.git
 eb465ee..9992344  main -> main

```

Рисунок 17 – Отправка изменений на удаленный репозиторий

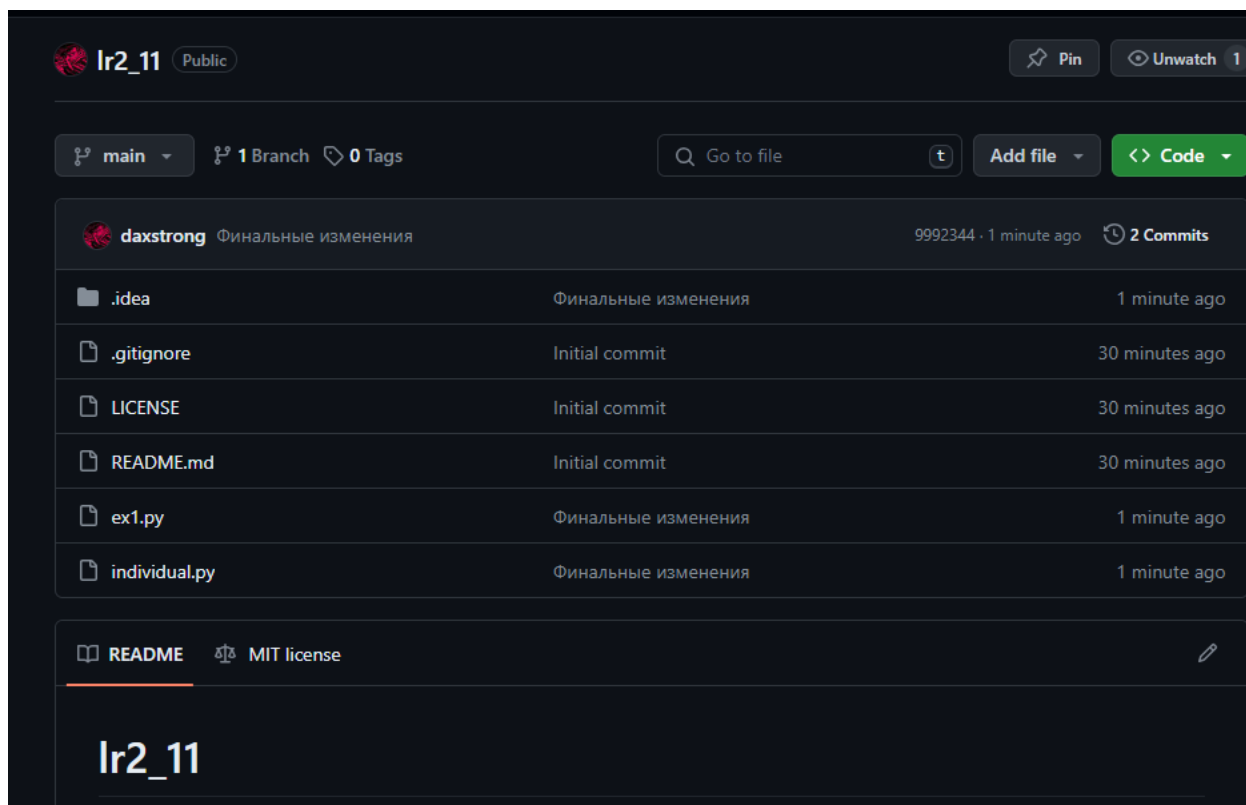


Рисунок 18 – Изменения удаленного репозитория

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.