

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ
ФЕДЕРАЦИИ**

**Федеральное государственное автономное образовательное учреждение
высшего образования**

«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт цифрового развития

Кафедра информационных систем и технологий

Отчет по лабораторной работе №14.

Дисциплина: «Основы программной инженерии»

Выполнил:

Студент группы ПИЖ-б-о-22-1,

направление подготовки: 09.03.04

«Программная инженерия»

ФИО: Гуртовой Ярослав Дмитриевич

Проверил:

Богданов С.С

Ставрополь 2023

Тема: Лабораторная работа 2.11 Замыкания в языке Python.

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

Выполнение работы:

1. Изучил теоретический материал работы.
2. Создал репозиторий на git.hub.

Owner * / Repository name *

KingItProgger / lr-2.11

✓ lr-2.11 is available.

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

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](#).

You are creating a public repository in your personal account.

Create repository

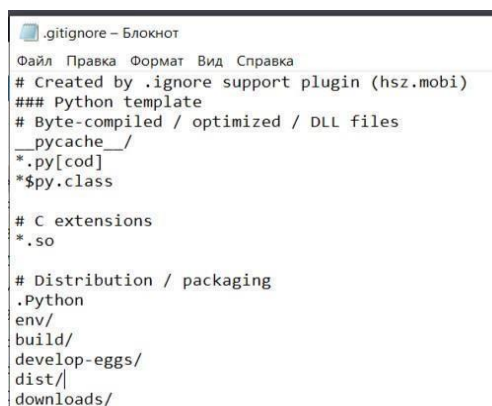
Рисунок 1 – создание репозитория

3. Клонировал репозиторий.

```
PS C:\Users\User\Desktop\учеба\Зсемак\змій\lr_2.11> git clone http
Cloning into 'lr-2.11'...
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.
PS C:\Users\User\Desktop\учеба\Зсемак\змій\lr_2.11>
```

Рисунок 2 – клонирование репозитория 4.

Дополнить файл gitignore необходимыми правилами.



```
.gitignore – Блокнот
Файл Правка Формат Вид Справка
# Created by .ignore support plugin (hsz.mobi)
### Python template
# Byte-compiled / optimized / DLL files
__pycache__/
*.py[cod]
*$py.class

# C extensions
*.so

# Distribution / packaging
.Python
env/
build/
develop-eggs/
dist/
downloads/
```

Рисунок 3 – .gitignore для IDE PyCharm

4. Организовать свой репозиторий в соответствии с моделью ветвления git-flow.

```
PS C:\Users\User\Desktop\учеба\Зсемак\змій\lr_2.11\lr-2.11> git checkout -b develop
Switched to a new branch 'develop'
```

Рисунок 4 – создание ветки develop

5. Проработал примеры из методички.

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

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

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

Рисунок 5 – пример 1

6. Используя замыкания функций, объявите внутреннюю функцию, которая принимает в качестве параметров фамилию и имя, а затем, заносит в шаблон эти данные. Сам шаблон – это строка, которая передается внешней функции и, например, может иметь такой вид: «Уважаемый %F%, %N%! Вы делаете работу по замыканиям функций.» Здесь %F% - это фрагмент куда нужно подставить фамилию, а %N% - фрагмент, куда нужно подставить имя. (Шаблон может быть и другим, вы это определяете сами). Здесь важно, чтобы внутренняя функция умела подставлять данные в шаблон, формировать новую строку и возвращать результат. Вызовите внутреннюю функцию замыкания и отобразите на экране результат ее работы.

```

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

def create_greeting_template(template):
    def inner_function(last_name, first_name):
        formatted_template = template.replace('%F%', last_name).replace('%N%', first_name)
        return formatted_template
    return inner_function

if __name__ == "__main__":
    n = input("Введите вашу фамилию: ")
    l = input("Введите ваше имя: ")

    # Создаем замыкание с шаблоном
    greeting_template = create_greeting_template("Уважаемый %F%, %N%! Вы делаете работу по замыканиям функций.")

    # Вызываем внутреннюю функцию замыкания и отображаем результат
    result = greeting_template(n, l)
    print(result)

```

Рисунок 6 – индивидуальное задание

```

C:\Users\User\AppData\Local\Programs\Py
Введите вашу фамилию: Гуртовой
Введите ваше имя: Ярослав
Уважаемый Гуртовой, Ярослав! Вы делаете
Process finished with exit code 0

```

Рисунок 7 – индивидуальное задание

7. Зафиксировал все изменения в github в ветке develop.

```

PS C:\Users\User\Desktop\учеба\Зсемак\змй\lr_2.11\lr-2.11> git add .
warning: in the working copy of 'pycharm/.idea/inspectionProfiles/profiles_settings.xml'
e next time Git touches it
PS C:\Users\User\Desktop\учеба\Зсемак\змй\lr_2.11\lr-2.11> git commit -m "laba"
[develop a5e1444] laba
7 files changed, 62 insertions(+)
create mode 100644 pycharm/.idea/.gitignore
create mode 100644 pycharm/.idea/inspectionProfiles/profiles_settings.xml
create mode 100644 pycharm/.idea/lr_2.11.iml
create mode 100644 pycharm/.idea/misc.xml
create mode 100644 pycharm/.idea/modules.xml
create mode 100644 pycharm/ex1.py
create mode 100644 pycharm/my_task.py
PS C:\Users\User\Desktop\учеба\Зсемак\змй\lr_2.11\lr-2.11>

```

Рисунок 8 – фиксация изменений в ветку develop

8. Слил ветки.

```
PS C:\Users\User\Desktop\учеба\3семак\эмий\lr_2.11\lr-2.11> git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
PS C:\Users\User\Desktop\учеба\3семак\эмий\lr_2.11\lr-2.11> git merge develop
Updating 5a20b5f..a5e1444
Fast-forward
 pycharm/.idea/.gitignore          | 3 +++
 .../.idea/inspectionProfiles/profiles_settings.xml | 6 ++++++
 pycharm/.idea/lr 2.11.iml         | 8 ++++++++
 pycharm/.idea/misc.xml            | 4 ++++
 pycharm/.idea/modules.xml         | 8 ++++++++
 pycharm/ex1.py                    | 14 ++++++++
 pycharm/my_task.py                | 19 ++++++++
7 files changed, 62 insertions(+)
create mode 100644 pycharm/.idea/.gitignore
create mode 100644 pycharm/.idea/inspectionProfiles/profiles_settings.xml
create mode 100644 pycharm/.idea/lr 2.11.iml
create mode 100644 pycharm/.idea/misc.xml
create mode 100644 pycharm/.idea/modules.xml
create mode 100644 pycharm/ex1.py
create mode 100644 pycharm/my_task.py
PS C:\Users\User\Desktop\учеба\3семак\эмий\lr_2.11\lr-2.11>
```

Рисунок 9 – сливание ветки develop в ветку main

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?

Замыкание (closure) в Python - это функция, которая запоминает значения в окружающей (внешней) области видимости, даже если эта область видимости больше не существует. Замыкание возникает, когда внутри функции определены вложенные функции, и вложенная функция ссылается на переменные из внешней функции.

3. Что подразумевает под собой область видимости Local?

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

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.

5. Что подразумевает под собой 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 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?

Использование замыканий в Python обычно связано с созданием функций внутри других функций и возвратом этих вложенных функций. Замыкания полезны, когда вам нужно передать часть контекста (переменные) в функцию, чтобы она могла использовать их даже после того, как внешняя функция завершила свою работу.

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

Замыкания в Python можно использовать для построения иерархических данных, таких как деревья или вложенные структуры. Замыкания позволяют сохранять состояние внешней функции внутри вложенной функции, что обеспечивает уровень иерархии.

