

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 №15
дисциплины «Основы программной инженерии»

Выполнил:
Магомедов Имран Борисович
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:
Воронкин Р.А., кандидат
технических наук, доцент кафедры
инфокоммуникаций

(подпись)

Отчет защищен с оценкой _____

Дата защиты _____

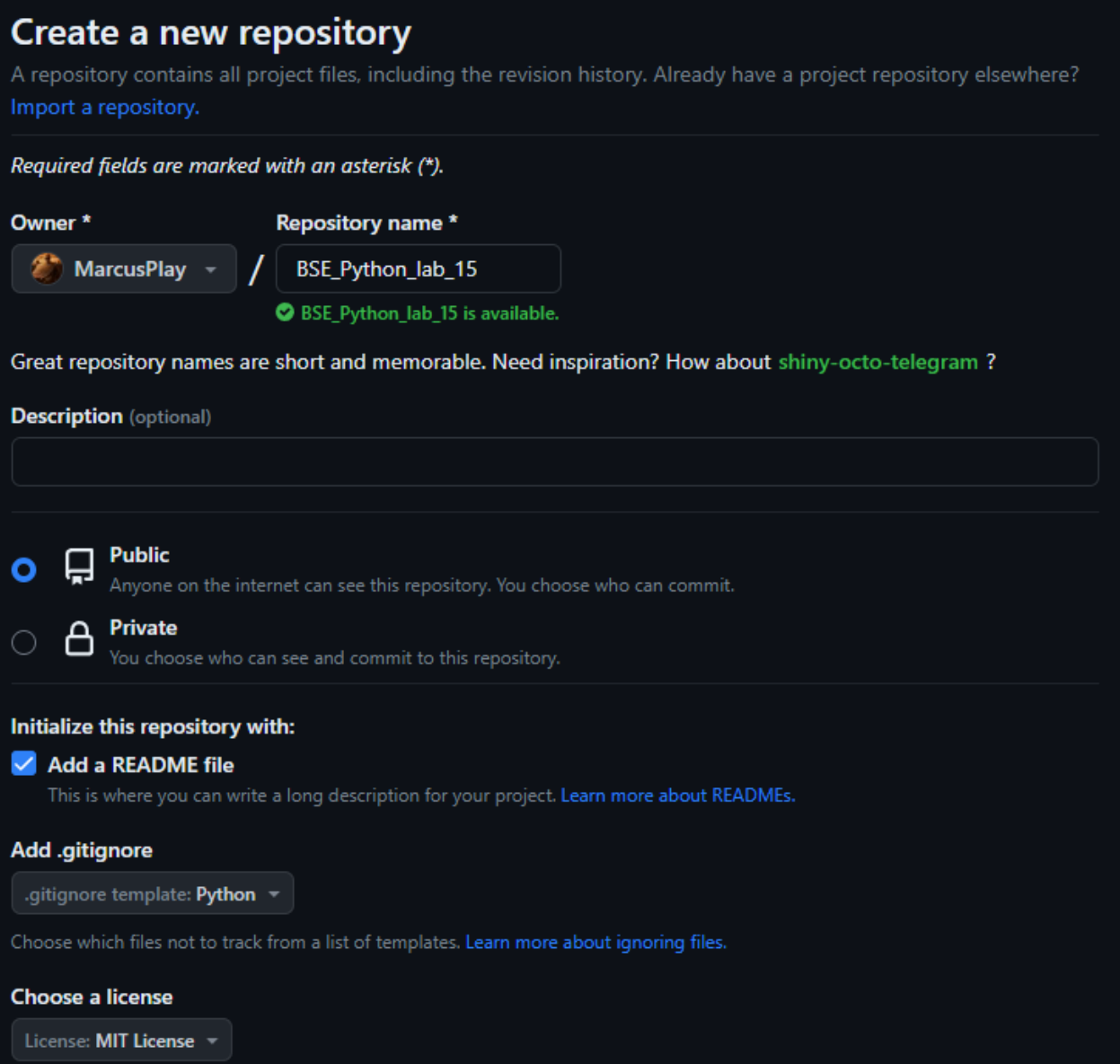
Ставрополь, 2023 г.

ТЕМА: Декораторы функций в языке Python

ЦЕЛЬ РАБОТА: приобретение навыков по работе с декораторами функций при написании программ с помощью языка программирования Python версии 3.x.

Методика и порядок выполнения работы

1. Изучить теоретический материал работы.
2. 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 ().*

Owner * MarcusPlay / **Repository name *** BSE_Python_lab_15

✔ BSE_Python_lab_15 is available.

Great repository names are short and memorable. Need inspiration? How about [shiny-octo-telegram](#) ?

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

3. Выполните клонирование созданного репозитория.

```
ImranāKaskad MINGW64 ~/Desktop/Лабораторная работа №15
$ git clone https://github.com/MarcusPlay/BSE_Python_lab_15.git
Cloning into 'BSE_Python_lab_15'...
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.
```

4. Организуйте свой репозиторий в соответствие с моделью ветвления git-flow.

```
ImranāKaskad MINGW64 ~/Desktop/Лабораторная работа №15/BSE_Python_lab_15 (main)
$ git checkout -b develop
Switched to a new branch 'develop'
```

5. Выполнить индивидуальное задание. Составить программу с использованием замыканий для решения задачи. Номер варианта 2. На вход программы поступает строка из целых чисел, записанных через пробел. Напишите функцию `get_list`, которая преобразовывает эту строку в список из целых чисел и возвращает его. Определите декоратор для этой функции, который сортирует список чисел, полученный из вызываемой в нем функции. Результат сортировки должен возвращаться при вызове декоратора. Вызовите декорированную функцию `get_list` и отобразите полученный отсортированный список на экране.

```
def sort_decorator(func):
    def wrapper(*args, **kwargs):
        result = func(*args, **kwargs)
        sorted_result = sorted(result)
        return sorted_result
    return wrapper

@sort_decorator
def get_list(input_string):
    number_list = list(map(int, input_string.split()))
    return number_list

if __name__=="__main__":
    input_string = input("Введите строку из целых чисел через пробел: ")
    sorted_list = get_list(input_string)
    print("Отсортированный список:", sorted_list)
```

```
Imran@Kaskad MINGW64 ~/Desktop/Лабораторная работа №15/BSE_Python_lab_15 (develop)
$ python3 individual_task.py
Введите строку из целых чисел через пробел: 5 2 8 1 7
отсортированный список: [1, 2, 5, 7, 8]
```

6. Зафиксируйте изменения в репозитории.

```
Imran@Kaskad MINGW64 ~/Desktop/Лабораторная работа №15/BSE_Python_lab_15 (develop)
$ git commit -m "added individual_task"
[develop c6eb51c] added individual_task
1 file changed, 31 insertions(+)
create mode 100644 individual_task.py
```

7. Добавьте отчет по лабораторной работе в *формате PDF* в папку *doc* репозитория. Зафиксируйте изменения.

8. Выполните слияние ветки для разработки с веткой *master/main*.

9. Отправьте сделанные изменения на сервер GitHub.

10. Отправьте адрес репозитория GitHub на электронный адрес преподавателя.

ВОПРОСЫ ДЛЯ ЗАЩИТЫ РАБОТЫ

1. Что такое декоратор?

Декоратор в Python — это специальный синтаксис, который позволяет изменять поведение функций или методов без изменения их кода. Декораторы

часто используются для добавления дополнительного функционала к существующим функциям.

2. Почему функции являются объектами первого класса?

Функции в Python считаются объектами первого класса, потому что они могут быть:

- Присвоены переменной.
- Переданы в качестве аргумента другой функции.
- Возвращены из функции.
- Хранены в структурах данных, таких как списки или словари.

3. Каково назначение функций высших порядков?

Функции высших порядков в Python могут принимать одну или несколько функций в качестве аргументов, возвращать функцию в качестве результата, или и то, и другое. Они позволяют более гибко и абстрактно работать с функциями, улучшая читаемость и повторное использование кода.

4. Как работают декораторы?

Декораторы в Python применяются к функции с использованием символа `@` перед определением функции. Декоратор — это функция, которая принимает другую функцию и возвращает новую функцию, изменяя или расширяя ее поведение. Декораторы применяются сверху вниз, то есть последний указанный декоратор будет применен первым.

5. Какова структура декоратора функций?

Пример структуры декоратора:

```
def my_decorator(func):  
    def wrapper():  
        print("Что-то происходит перед вызовом  
функции.")  
        func()  
        print("Что-то происходит после вызова  
функции.")  
    return wrapper
```

```
@my_decorator
def say_hello():
    print("Привет!")

say_hello()
```

В данном примере `my_decorator` — это декоратор, который добавляет дополнительное поведение вокруг функции `say_hello`.

6. Самостоятельно изучить как можно передать параметры декоратору, а не декорируемой функции?

Декоратор может принимать параметры, что позволяет настраивать его поведение. Для этого можно создать дополнительную функцию, которая возвращает сам декоратор. Пример:

```
def parametrized_decorator(param):
    def decorator(func):
        def wrapper():
            print(f"Декоратор с параметром {param}")
            func()
        return wrapper
    return decorator

@parametrized_decorator("пример")
def my_function():
    print("Исходная функция")

my_function()
```

В этом примере `parametrized_decorator` — это декоратор, принимающий параметр и возвращающий фактический декоратор, который, в свою очередь, применяется к функции `my_function`.