

**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 №15.

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

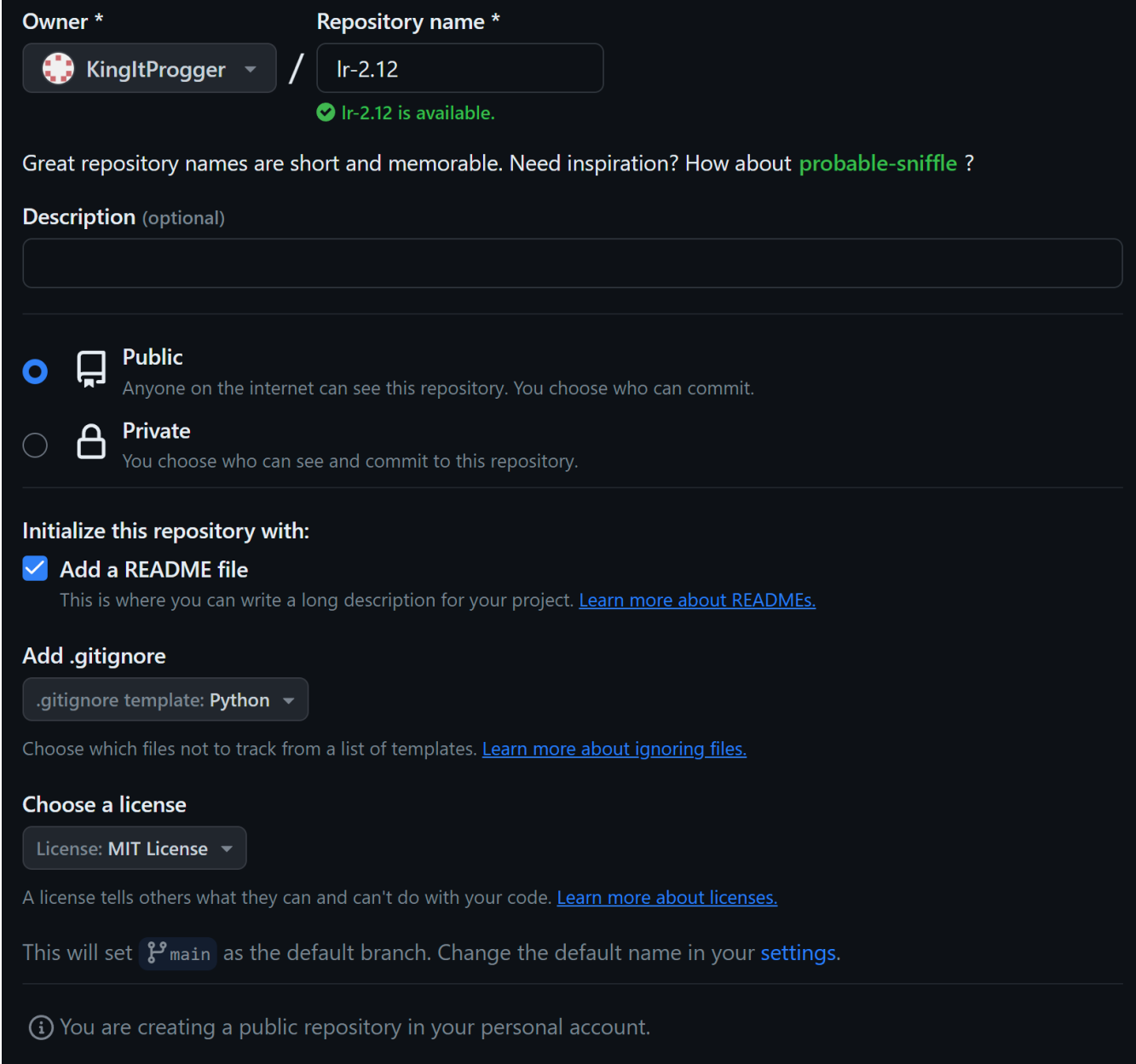
Ставрополь 2023

Тема: Лабораторная работа 2.12 Декораторы функций в языке Python.

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

Выполнение работы:

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



The screenshot shows the GitHub 'Create new repository' page. At the top, the 'Owner' is set to 'KingItProgger' and the 'Repository name' is 'lr-2.12'. A green checkmark indicates 'lr-2.12 is available.' Below this, a message says 'Great repository names are short and memorable. Need inspiration? How about **probable-sniffle** ?'. The 'Description' field is empty. Under 'Visibility', the 'Public' option is selected with a radio button, and the 'Private' option is unselected. The 'Initialize this repository with:' section has 'Add a README file' checked. Below that, the '.gitignore template' is set to 'Python'. The 'Choose a license' section has 'MIT License' selected. At the bottom, a note states 'This will set `main` as the default branch. Change the default name in your [settings](#).' and a footer note says 'You are creating a public repository in your personal account.'

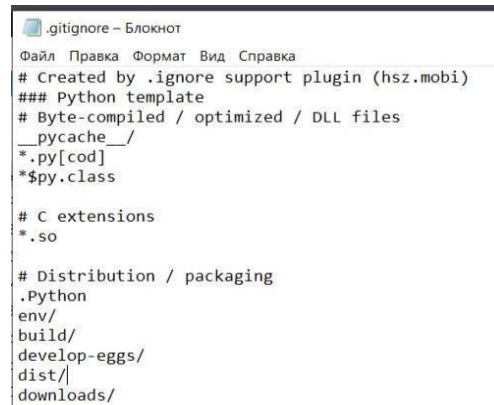
Рисунок 1 – создание репозитория

3. Клонировал репозиторий.

```
PS C:\Users\User\Desktop\учеба\Зсемак\змії\lr_2.12> git clone
Cloning into 'lr-2.12'...
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\змії\lr_2.12>
```

Рисунок 2 – клонирование репозитория 4.

Дополнить файл gitignore необходимыми правилами.



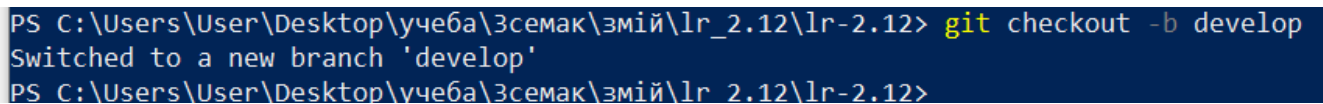
```
.gitignore – Блокнот
Файл Правка Формат Вид Справка
# Created by .ignore support plugin (hsz.mobi)
### Python template
# Byte-compiled / optimized / DLL files
__pycache__/
*.py[cod]
*$py.class

# C extensions
*.so

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

Рисунок 3 – .gitignore для IDE PyCharm

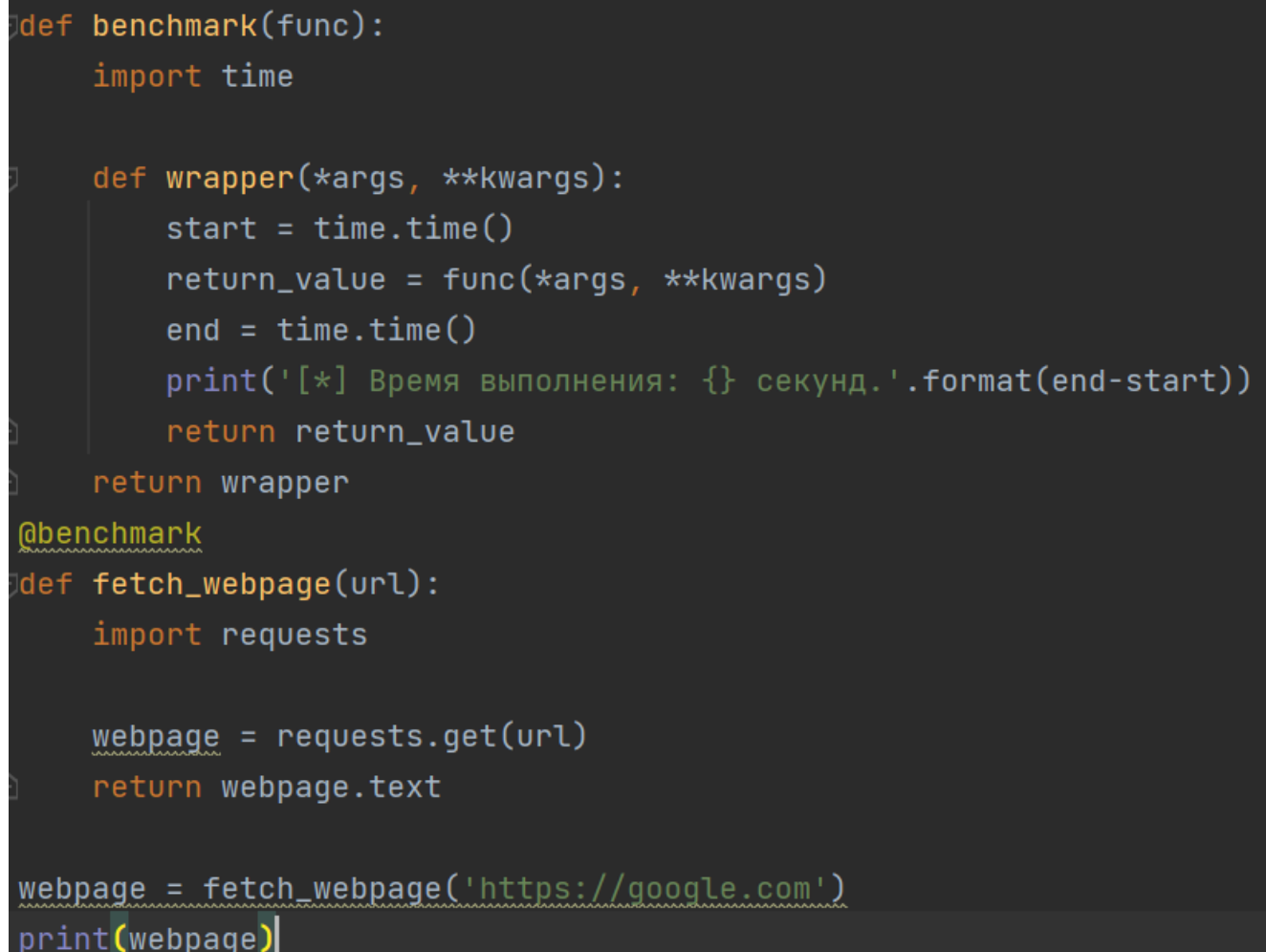
4. Организовать свой репозиторий в соответствии с моделью ветвления git-flow.



```
PS C:\Users\User\Desktop\учеба\Зсемак\змій\lr_2.12\lr-2.12> git checkout -b develop
Switched to a new branch 'develop'
PS C:\Users\User\Desktop\учеба\Зсемак\змій\lr_2.12\lr-2.12>
```

Рисунок 4 – создание ветки develop

5. Проработал примеры из методички.



```
def benchmark(func):
    import time

    def wrapper(*args, **kwargs):
        start = time.time()
        return_value = func(*args, **kwargs)
        end = time.time()
        print('[*] Время выполнения: {} секунд.'.format(end-start))
        return return_value

    return wrapper

@benchmark
def fetch_webpage(url):
    import requests

    webpage = requests.get(url)
    return webpage.text

webpage = fetch_webpage('https://google.com')
print(webpage)
```

Рисунок 5 – пример 1

6. Вводится строка целых чисел через пробел. Напишите функцию, которая преобразовывает эту строку в список чисел и возвращает их сумму. Определите декоратор для этой функции, который имеет один параметр start – начальное значение суммы. Примените декоратор со значением start=5 к функции и вызовите декорированную функцию. Результат отобразите на экране.

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

def sum_decorator(start):
    def decorator(func):
        def wrapper(input_string):
            numbers = [int(num) for num in input_string.split()]
            result = func(numbers)
            return result + start
        return wrapper
    return decorator

@sum_decorator(start=5)
def sum_of_numbers(numbers):
    return sum(numbers)

if __name__ == "__main__":
    # Ввод строки целых чисел через пробел
    input_string = input("Введите строку целых чисел через пробел: ")

    # Вызов декорированной функции и вывод результата
    result = sum_of_numbers(input_string)
    print(f"Сумма чисел с учетом начального значения: {result}")
```

Рисунок 6 – индивидуальное задание

```
C:\Users\User\AppData\Local\Programs\Python\Python
Введите строку целых чисел через пробел: 1 2 3 4
Сумма чисел с учетом начального значения: 15
```

Рисунок 7 – индивидуальное задание

7. Зафиксировал все изменения в github в ветке develop.

```

PS C:\Users\User\Desktop\учеба\Зсемак\эмий\lr_2.12\lr-2.12> 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\эмий\lr_2.12\lr-2.12> git commit -m "laba"
[develop 8b9e6e5] laba
 7 files changed, 71 insertions(+)
 create mode 100644 pycharm/.idea/.gitignore
 create mode 100644 pycharm/.idea/inspectionProfiles/profiles_settings.xml
 create mode 100644 pycharm/.idea/lr 2.12.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\эмий\lr_2.12\lr-2.12>

```

Рисунок 8 – фиксация изменений в ветку develop

8. Слил ветки.

```

PS C:\Users\User\Desktop\учеба\Зсемак\эмий\lr_2.12\lr-2.12> git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
PS C:\Users\User\Desktop\учеба\Зсемак\эмий\lr_2.12\lr-2.12> git merge develop
Updating 58ce72b..8b9e6e5
Fast-forward
 pycharm/.idea/.gitignore          | 3 +++
 .../.idea/inspectionProfiles/profiles_settings.xml | 6 ++++++
 pycharm/.idea/lr 2.12.iml         | 8 ++++++++
 pycharm/.idea/misc.xml            | 4 +++++
 pycharm/.idea/modules.xml         | 8 ++++++++
 pycharm/ex1.py                    | 19 +++++++++++++++++++++
 pycharm/my_task.py                | 23 ++++++++++++++++++++++
 7 files changed, 71 insertions(+)
 create mode 100644 pycharm/.idea/.gitignore
 create mode 100644 pycharm/.idea/inspectionProfiles/profiles_settings.xml
 create mode 100644 pycharm/.idea/lr 2.12.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\эмий\lr_2.12\lr-2.12>

```

Рисунок 9 – сливание ветки develop в ветку main

Контрольные вопросы:

1. Что такое декоратор?

Декораторы — один из самых полезных инструментов в Python, однако новичкам они могут показаться непонятными. Декоратор — это функция, которая позволяет обернуть другую функцию для расширения её функциональности без непосредственного изменения её кода

2. Почему функции являются объектами первого класса?

Объектами первого класса в контексте конкретного языка программирования называются элементы, с которыми можно делать всё то же, что и с любым другим объектом: передавать как параметр, возвращать из функции

и присваивать переменной. Именно поэтому функции являются объектами первого класса.

3. Каково назначение функций высших порядков?

Он принимает на входе функцию и возвращает другую функцию, производную от исходной. Функции высших порядков в программировании работают точно так же — они либо принимают функцию(и) на входе и/или возвращают функцию(и).

4. Как работают декораторы?

Декораторы в Python представляют собой способ изменить поведение функции или метода, обернув его в другую функцию. Это мощный механизм, который позволяет добавлять или изменять функциональность функций без изменения их кода. Декораторы часто используются для внесения дополнительной логики, проверок или изменений в функции.

Декораторы позволяют модифицировать поведение функций или методов, делая код более модульным и легким для понимания. Они часто используются, например, для логирования, обработки ошибок, кеширования, аутентификации и других аспектов функциональности программы.

5. Какова структура декоратора функций?

```
def decorator_function(original_function):  
    def wrapper_function(*args, **kwargs):  
        # Дополнительный код, выполняемый перед вызовом оригинальной  
функции  
        result = original_function(*args, **kwargs)  
        # Дополнительный код, выполняемый после вызова оригинальной  
функции  
        return result  
    return wrapper_function
```

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

Использование функций-фабрик декораторов:

```
def decorator_factory(param):
    def decorator_function(original_function):
        def wrapper_function(*args, **kwargs):
            print(f"Дополнительный код с параметром {param} перед вызовом функции")
            result = original_function(*args, **kwargs)
            print(f"Дополнительный код с параметром {param} после вызова функции")
            return result
        return wrapper_function
    return decorator_function

# Использование декоратора с параметром
@decorator_factory(param="some_parameter")
def example_function():
    print("Оригинальная функция")

# Вызов функции, обернутой в декоратор
example_function()
```

2. Использование частичного применения (functools.partial):

```
from functools import partial

def decorator_function(param, original_function, *args, **kwargs):
    print(f"Дополнительный код с параметром {param} перед вызовом функции")
    result = original_function(*args, **kwargs)
    print(f"Дополнительный код с параметром {param} после вызова функции")
    return result

# Создание частичной функции с фиксированным параметром
decorator_with_param = partial(decorator_function, param="some_parameter")

# Использование декоратора с параметром
@decorator_with_param
def example_function():
    print("Оригинальная функция")

# Вызов функции, обернутой в декоратор
example_function()
```