

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙ-
СКОЙ ФЕДЕРАЦИИ**

**Федеральное государственное автономное
образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

Кафедра инфокоммуникаций

Основы кроссплатформенного программирования

Отчет по лабораторной работе №2.12

Тема: «Декораторы функций в языке Python»

Выполнил студент группы

ИВТ-б-о-21-1

Богадунов В.И. « » _____ 20__ г.

Подпись студента _____

Работа защищена « » _____ 20__ г.

Проверил доцент

Кафедры инфокоммуникаций, старший
преподаватель

Воронкин Р.А.

(подпись)

Ставрополь 2022

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

Ход работы:

1. Создал репозиторий в GitHub, дополнил правила в .gitignore для работы с IDE PyCharm с ЯП Python, выбрал лицензию MIT, клонировал его на компьютер и организовал в соответствии с моделью ветвления git-flow.

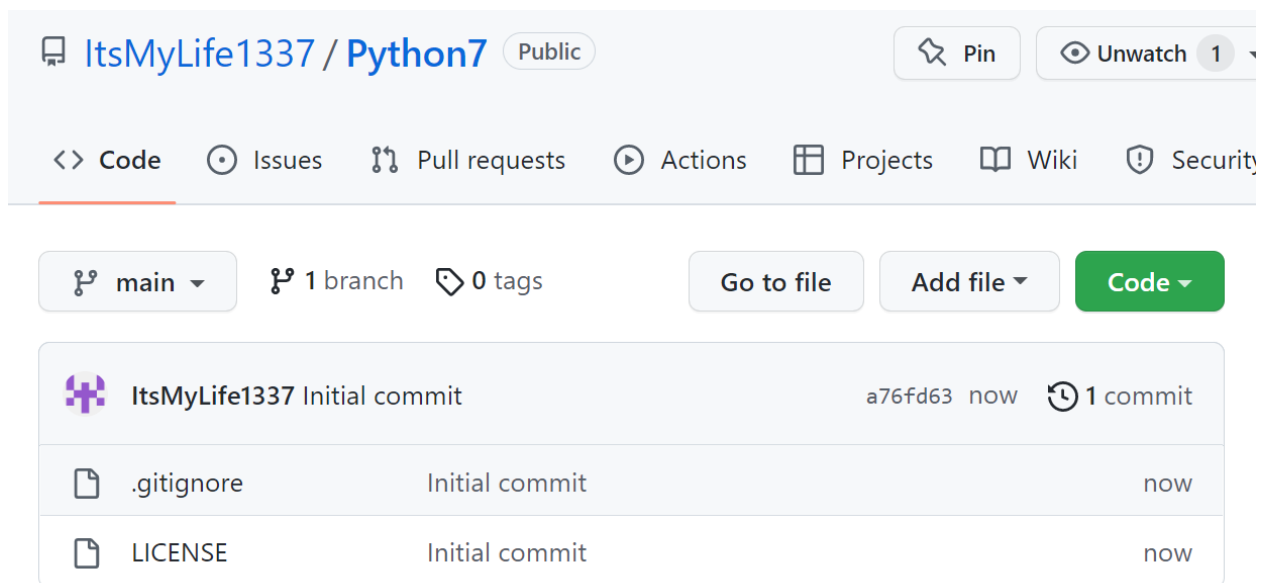


Рисунок 1.1 – Созданный репозиторий

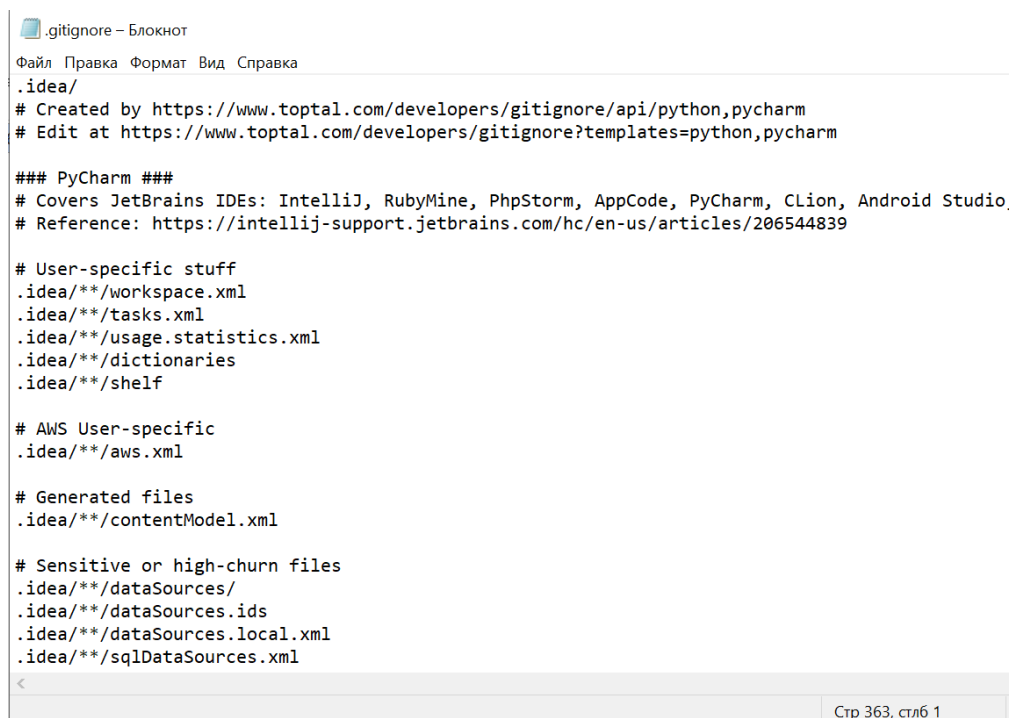


Рисунок 1.2 – Дополнил правила в .gitignore

```

c:\Users\Admin\Desktop\git\Python7>git flow init

Which branch should be used for bringing forth production releases?
- main
Branch name for production releases: [main]
Branch name for "next release" development: [develop]

How to name your supporting branch prefixes?
Feature branches? [feature/]
Bugfix branches? [bugfix/]
Release branches? [release/]
Hotfix branches? [hotfix/]
Support branches? [support/]
Version tag prefix? []
Hooks and filters directory? [C:/Users/Admin/Desktop/git/Python7/.git/hooks]

c:\Users\Admin\Desktop\git\Python7>

```

Рисунок 1.3 – Организация репозитория в соответствии с моделью ветвления git-flow

2. Создал проект Pycharm в папке репозитория, проработал примеры ЛР.

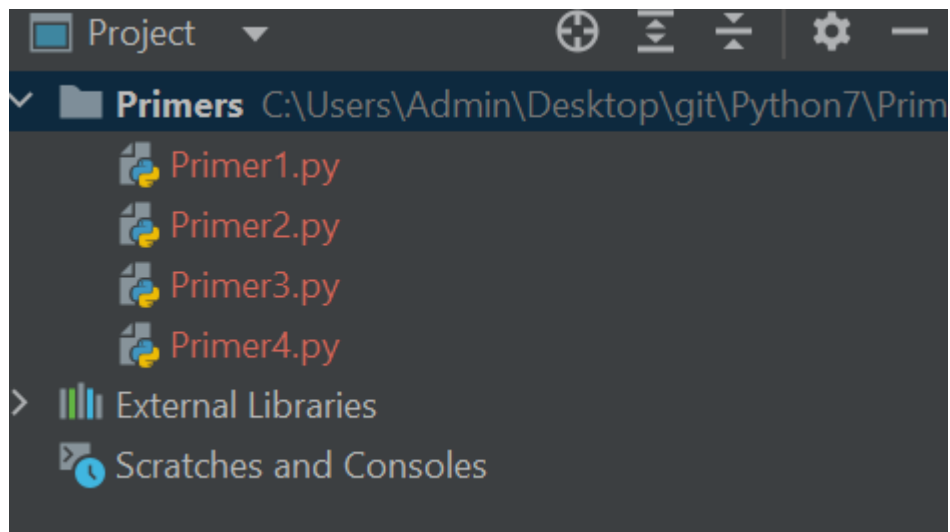


Рисунок 2.1 – Созданные проекты

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

def wrapper_function():
    def hello_world():
        print('Hello world!')
    hello_world()

if __name__ == "__main__":
    wrapper_function()
```

Рисунок 2.2 – Результат выполнения примера №1

```

def func():
    print('Выходим из обёртки')
    return wrapper

@decorator_function
def hello_world():
    print('Hello world!')

if __name__ == "__main__":
    hello_world()

```

Primer2 x

C:\Users\Admin\AppData\Local\Programs\Python\Python39\python.exe C

Функция-обёртка!

Оборачиваемая функция: <function hello_world at 0x000002C934C75160>

Выполняем обёрнутую функцию...

Hello world!

Выходим из обёртки

Process finished with exit code 0

Рисунок 2.3 – Результат выполнения примера №2

```
return wrapper

@benchmark
def fetch_webpage():
    import requests
    requests.get('https://google.com')

if __name__ == "__main__":
    fetch_webpage()

if __name__ == "__main__":
    fetch_webpage()

Primer3 x
C:\Users\Admin\AppData\Local\Programs\Python\Python38\python.exe
[*] Время выполнения: 1.9278357028961182 секунд.

Process finished with exit code 0
```

Рисунок 2.4 – Результат выполнения примера №3

```
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)
    fetch_webpage()

Primer4 x
var a=window.innerWidth,b=window.innerHeight;if(!a||!b){var c=window.innerWidth,d=this||self,e=function(a){return a};var g;var l=function(a,b){this.g=b===h?a:"";l.prototype.toString=function p(a){google.timers&&google.timers.load&&google.tick&&google.F_installCss(c)}(function(){google.jl={blt:'none',chnk:0,dw:false,dwu:true,emtn:0,e

Process finished with exit code 0
```

Рисунок 2.5 – Результат выполнения примера №4

Индивидуальное задание. В – 1. Объявите функцию с именем `get_sq`, которая вычисляет площадь прямоугольника по двум параметрам: `width` и `height` – ширина и высота прямоугольника и возвращает результат. Определите декоратор для этой функции с именем (внешней функции) `func_show`, который отображает результат на экране в виде строки (без кавычек): "Площадь прямоугольника: <значение>". Вызовите декорированную функцию `get_sq`.

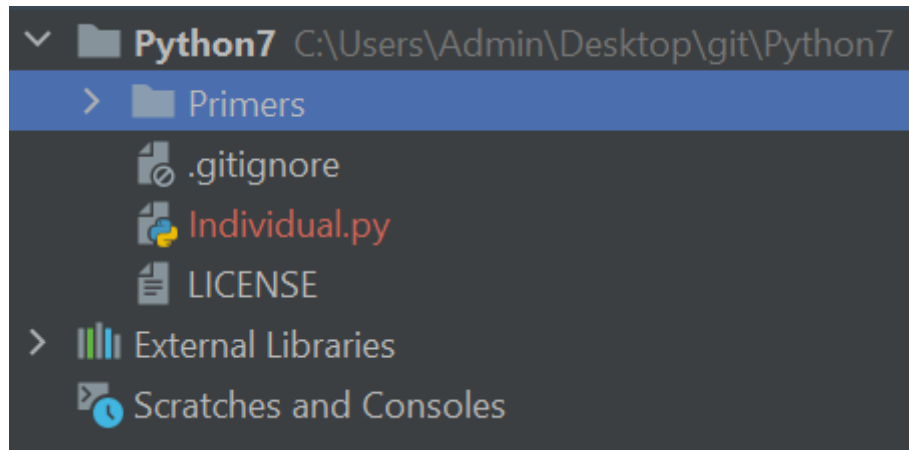


Рисунок 3.1 – Созданный проект

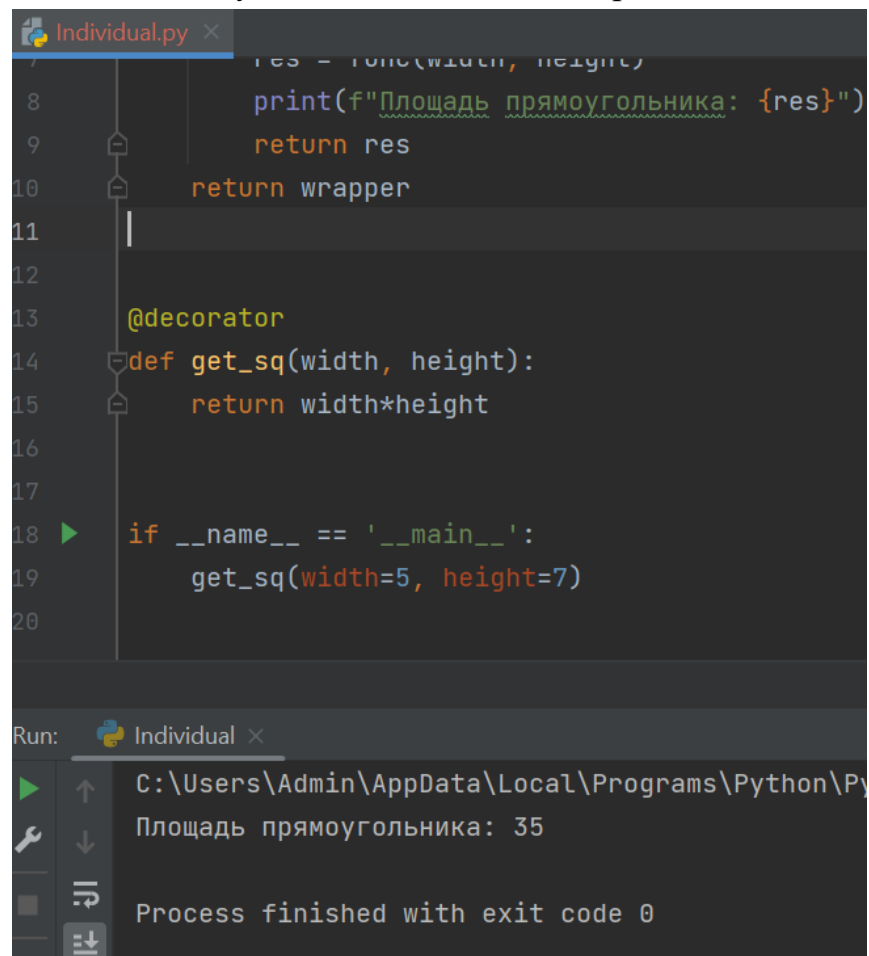


Рисунок 3.2 – Результат выполнения программы

Вывод: в результате выполнения лабораторной работы были приобретены практические навыки и теоретические сведения по работе с декораторами функций при написании программ с помощью языка программирования Python версии 3.x..

Ответы на контрольные вопросы:

1. Что такое декоратор?

Декоратор — это функция, которая позволяет обернуть другую функцию для расширения её функциональности без непосредственного изменения её кода. Вот почему декораторы можно рассматривать как практику метапрограммирования, когда программы могут работать с другими программами как со своими данными.

2. Почему функции являются объектами первого класса?

В Python всё является объектом, а не только объекты, которые вы создаёте из классов. В этом смысле он (Python) полностью соответствует идеям объектно-ориентированного программирования. Это значит, что в Python всё это — объекты:

- числа;
- строки;
- классы (да, даже классы!);
- функции (то, что нас интересует).

Тот факт, что всё является объектами, открывает перед нами множество возможностей. **Мы можем сохранять функции в переменные, передавать их в качестве аргументов и возвращать из других функций. Можно даже определить одну функцию внутри другой. Иными словами, функции — это объекты первого класса.**

3. Каково назначение функций высших порядков?

Функции высших порядков — это такие функции, которые могут принимать в качестве аргументов и возвращать другие функции.

4. Как работают декораторы?

```
def decorator_function(func):  
  
    def wrapper():  
  
        print('Функция-обёртка!')  
  
        print('Оборачиваемая функция: {}'.format(func))  
  
        print('Выполняем обёрнутую функцию...')  
  
        func()  
  
        print('Выходим из обёртки')  
  
    return wrapper
```

Здесь `decorator_function()` является функцией-декоратором. Как вы могли заметить, она является функцией высшего порядка, так как принимает функцию в качестве аргумента, а также возвращает функцию. Внутри `decorator_function()` мы определили другую функцию, обёртку, так сказать, которая обёртывает функцию-аргумент и затем изменяет её поведение. Декоратор возвращает эту обёртку.

5. Какова структура декоратора функций?

В 4 вопросе пример. Здесь **`decorator_function()` является функцией-декоратором.** Как вы могли заметить, она является функцией высшего порядка, так как принимает функцию в качестве аргумента, а также возвращает функцию. Внутри `decorator_function()` мы определили другую функцию, обёртку, так сказать, которая обёртывает функцию-аргумент и затем изменяет её поведение. Декоратор возвращает эту обёртку.

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

В декоратор можно передать и сам параметр. В этом случае нужно добавить ещё один слой абстракции, то есть – ещё одну функцию-обёртку. Это обязательно, поскольку аргумент передаётся декоратору. Затем, функция, которая вернулась, используется для декорации нужной.