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 №2.12**

**дисциплины «Программирование на Python»**

|  |
| --- |
| Выполнил:  Кенесбаев Хилол Куат улы  2 курс, группа ИВТ-б-о-22-1,  09.03.01 «Информатика и вычислительная техника», направленность (профиль)  «Программное обеспечение средств вычислительной техники и автоматизированных систем», очная форма обучения  (подпись) |
| Руководитель практики: Воронкин Р.А., доцент кафедры инфокоммуникаций  (подпись) |

Отчет защищен с оценкой Дата защиты

Ставрополь, 2023 г.

**Тема:** Декораторы функций в языке Python

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

**Порядок выполнения работы:**

1. Создал новый репозиторий, клонировал его, в нем создал ветку developer и перешел на нее.
2. Проработал примеры лабораторной работы:

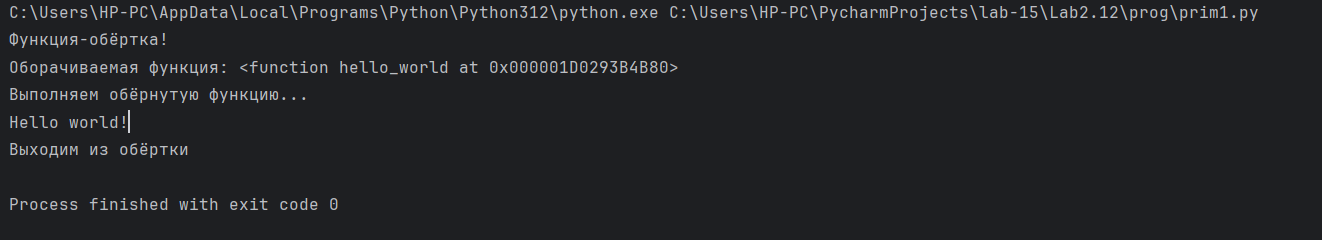


Рисунок 1 Вывод примера 1

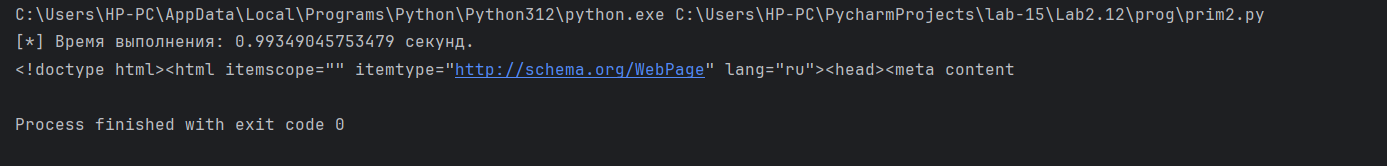


Рисунок 2 Вывод примера 2

1. Выполнил индивидуальное задание вариант 10: Объявите функцию, которая принимает строку, удаляет из нее все подряд идущие пробелы и переводит ее в нижний регистр – малые буквы. Результат (строка) возвращается функцией. Определите декоратор, который строку, возвращенную функцией, переводит в азбуку Морзе, используя следующий словарь для замены русских букв и символа пробела на соответствующие последовательности из точек и тире. Преобразованная строка возвращается декоратором. Примените декоратор к функции и вызовите декорарированную функцию. Результат отобразите на экране.

**Код программы:**

#!/usr/bin/env python3 # -\*- coding: utf-8 -\*- def pink(func):

def pnk(text, chars=" !?"):

print(text)

h = ''.join(map(lambda x: x if x not in chars else '-', func(text))) print(h)

while '--' in h:

h = h.replace('--', '-') print(h)

return h return pnk @pink

def wrapper(text):

p = text.lower()

t = {'ё': 'yo', 'а': 'a', 'б': 'b', 'в': 'v', 'г': 'g', 'д': 'd', 'е': 'e',

'ж': 'zh', 'з': 'z', 'и': 'i', 'й': 'y', 'к': 'k', 'л': 'l',

'м': 'm', 'н': 'n', 'о': 'o', 'п': 'p', 'р': 'r', 'с': 's', 'т': 't',

'у': 'u', 'ф': 'f', 'х': 'h', 'ц': 'c', 'ч': 'ch', 'ш': 'sh',

'щ': 'shch', 'ъ': '', 'ы': 'y', 'ь': '', 'э': 'e', 'ю': 'yu',

'я': 'ya'}

return p.translate({ord(key): t[key] for key in t}) if name == " main ":

s = 'Привет я Хилол' x = wrapper(s)

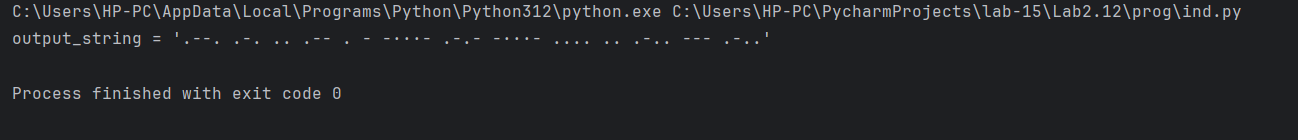


Рисунок 3 Вывод программы ind

**Ответы на контрольные вопросы:**

1. Что такое декоратор?

Декоратор  это функция, которая позволяет обернуть другуюфункцию для расширения её функциональности без непосредственного изменения её кода. Декораторы можно рассматривать как практику метапрограммирования, когда программы могут работать с другими

программами как со своими данными.

1. 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. С функцией все это делать можно, поэтому ее и можно назвать объектом первого класса.

1. Каково назначение функций высших порядков?

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

1. Как работают декораторы? Пример:

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. Она является функцией высшего порядка, так как принимает функцию в качестве аргумента, а также возвращает функцию. Внутри decorator\_function() опредена другая функция, которая обёртывает функцию-аргумент и затем изменяет её поведение. Декоратор возвращает эту обёртку.

Перед функцией остается просписать @decorator\_function.

Однако выражение с @ является всего лишь синтаксическим сахаром для hello\_world = decorator\_function(hello\_world).

1. Какова структура декоратора функций?

def decorator(func):

def wrapper(\*args, \*\*kwargs):

# Код до вызова целевой функции result = func(\*args, \*\*kwargs)

# Вызов целевой функции

# Код после вызова целевой функции return result

return wrapper

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

В Python можно передавать параметры декоратору, добавляя еще один уровень вложенности.

Напиример:

def decorator\_with\_parameters(param1, param2):

def actual\_decorator(func):

def wrapper(\*args, \*\*kwargs):

print(f"Decorator parameters: {param1}, {param2}") result = func(\*args, \*\*kwargs)

return resultreturn wrapper return actual\_decorator

Вызов декоратора с параметрами будет выглядеть так: @decorator\_with\_parameters(p1, p2)

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