МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
ИНСТИТУТ ЦИФРОВОГО РАЗВИТИЯ

Отчет по лабораторной работе №15 по дисциплине:

основы программной инженерии

Выполнила:

студент группы ПИЖ-б-о-20-1

Лазарева Дарья Олеговна

Проверил:

доцент кафедры инфокоммуникаций

Романкин Р.А.

Ставрополь, 2021 г.

Ход работы:

1. Выполнение примера с областью видимости local

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

2. Пример работы с областью видимости enclosing

```
>>> def hello_world():
...     print('Hello world!')
...
>>> type(hello_world)
<class 'function'>
>>> class Hello:
...     pass
...
>>> type(Hello)
<class 'type'>
>>> type(10)
<class 'int'>
>>>
```

3. Определение функции внутри других функций

```
>>> def wrapper_function():
...     def hello_world():
...         print('Hello world!')
...     hello_world()
...
>>> wrapper_function()
Hello world!
```

4. Передача функции в качестве аргументов

```
>>> def higher_order(func):
...     print('Получена функция {} в качестве аргумента'.format(func))
...     func()
...     return func
...
>>> higher_order(hello_world)
Получена функция <function hello_world at 0x000001DA1C2EA170> в качестве аргумента
Hello world!
<function hello_world at 0x000001DA1C2EA170>
```

5. Работа декоратора

```
>>> 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
```

6. Результат применения функции-обертки

```
>>> @decorator_function
... def hello_world():
...     print('Hello world!')
...
>>> hello_world()
Функция-обёртка!
Оборачиваемая функция: <function hello_world at 0x000001DA1C2EA4D0>
Выполняем обёрнутую функцию...
Hello world!
Выходим из обёртки
```

7. Выполнение 1 примера

```python
def benchmark(func):
    import time

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

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

fetch_webpage()
```

```
modul
C:\Users\79616\anaconda3\python.exe D:/УЧЕБА/ОПИ/15/modul.py
[*] Время выполнения: 1.6110856533050537 секунд.
```

8. Выполнение примера 2

```python
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')
```

```
modul1
C:\Users\79616\anaconda3\python.exe D:/УЧЕБА/ОПИ/15/modul1.py
[*] Время выполнения: 1.5091538429260254 секунд.
<!doctype html><html itemscope="" itemtype="http://schema.org/WebPage" lang="ru"><head><meta content="&#1055;&#1086;&#1080;
var f=this||self;var h,k=[];function l(a){for(var b;a&&(!a.getAttribute||!(b=a.getAttribute("eid")));)a=a.parentNode;return
function n(a,b,c,d,g){var e="";c||-1!==b.search("&ei=")||(e="&ei="+l(d),-1===b.search("&lei=")&&(d=m(d))&&(e+="&lei="+d));
google.y={};google.sy=[];google.x=function(a,b){if(a)var c=a.id;else{do c=Math.random();while(google.y[c])}google.y[c]=[a,
document.documentElement.addEventListener("submit",function(b){var a;if(a=b.target){var c=a.getAttribute("data-submitfalse"
</style><style>body,td,a,p,.h{font-family:arial,sans-serif}body{margin:0;overflow-y:scroll}#gog{padding:3px 8px 0}td{line-
var f=this||self;var g,h,k=null!==(g=f.mei)&&void 0!==g?g:1,l=null!==(h=f.sdo)&&void 0!==h?h:!0,p=0,q,r=google.erd,u=r.jsr
e);var n=a.fileName;n&&(b+="&script="+c(n),e&&n===window.location.href&&(e=document.documentElement.outerHTML.split("\n")[e
if (!iesg){document.f&&document.f.q.focus();document.gbqf&&document.gbqf.q.focus();}
```

Выполнение индивидуального задания. Вариант 3

Вводятся два списка (каждый с новой строки) из слов, записанных через пробел. Имеется функция, которая преобразовывает эти две строки в два списка слов и возвращает эти списки. Определите декоратор для этой функции, который из этих двух списков формирует словарь, в котором ключами являются слова из первого списка, а значениями – соответствующие элементы из второго списка. Полученный словарь должен возвращаться при вызове декоратора. Примените декоратор к первой функции и вызовите ее. Результат (словарь) отобразите на экране.

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

def decorator(func):
    def decorator_inside(A, B):
        data = func(A, B)
        return dict(zip(*data))

    return decorator_inside

@decorator
def listing(A, B):
    return A.split(), B.split()


if __name__ == '__main__':
    First = input("Введите первую строку: ")
    Second = input("Введите вторую строку: ")
    print(listing(First, Second))
```

```
Введите первую строку: 1 2 3
Введите вторую строку: -первый -второй -третий
{'1': '-первый', '2': '-второй', '3': '-третий'}
```

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.

2. Почему функции являются объектами первого класса?

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

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?

Декоратор – это функция, которая позволяет обернуть другую функцию для расширения её функциональности без непосредственного изменения её кода. Внутри декораторы мы определяем другую функцию, обёртку, так сказать, которая обёртывает функцию-аргумент и затем изменяет её поведение. Мы создаём декоратор, замеряющий время выполнения функции. Далее мы используем его функции, которая делает GET- запрос к главной странице. Чтобы измерить скорость, мы сначала сохраняем время перед выполнением обёрнутой функции, выполняем её снова сохраняем текущее время и вычитаем из него начальное.

Выражение @decorator_function вызывает decorator_function() с hello_world в качестве аргумента и присваивает имени hello_world возвращаемую функцию.

```
def benchmark(func):
    import time

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

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

fetch_webpage()
```

5. Какова структура декоратора функций?

```python
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)
```

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

```python
import functools


def decoration(*args):
    def dec(func):
        @functools.wraps(func)
        def decor():
            func()
            print(*args)
        return decor
    return dec


@decoration('This is *args')
def func_ex():
    print('Look at that')

if __name__ == '__main__':
    func_ex()
```

```
Look at that
This is *args

Process finished with exit code 0
```