

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 «Программирование на Python»

Выполнил:
Данилецкий Дмитрий Витальевич
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., канд. технических
наук, доцент кафедры
инфокоммуникаций

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

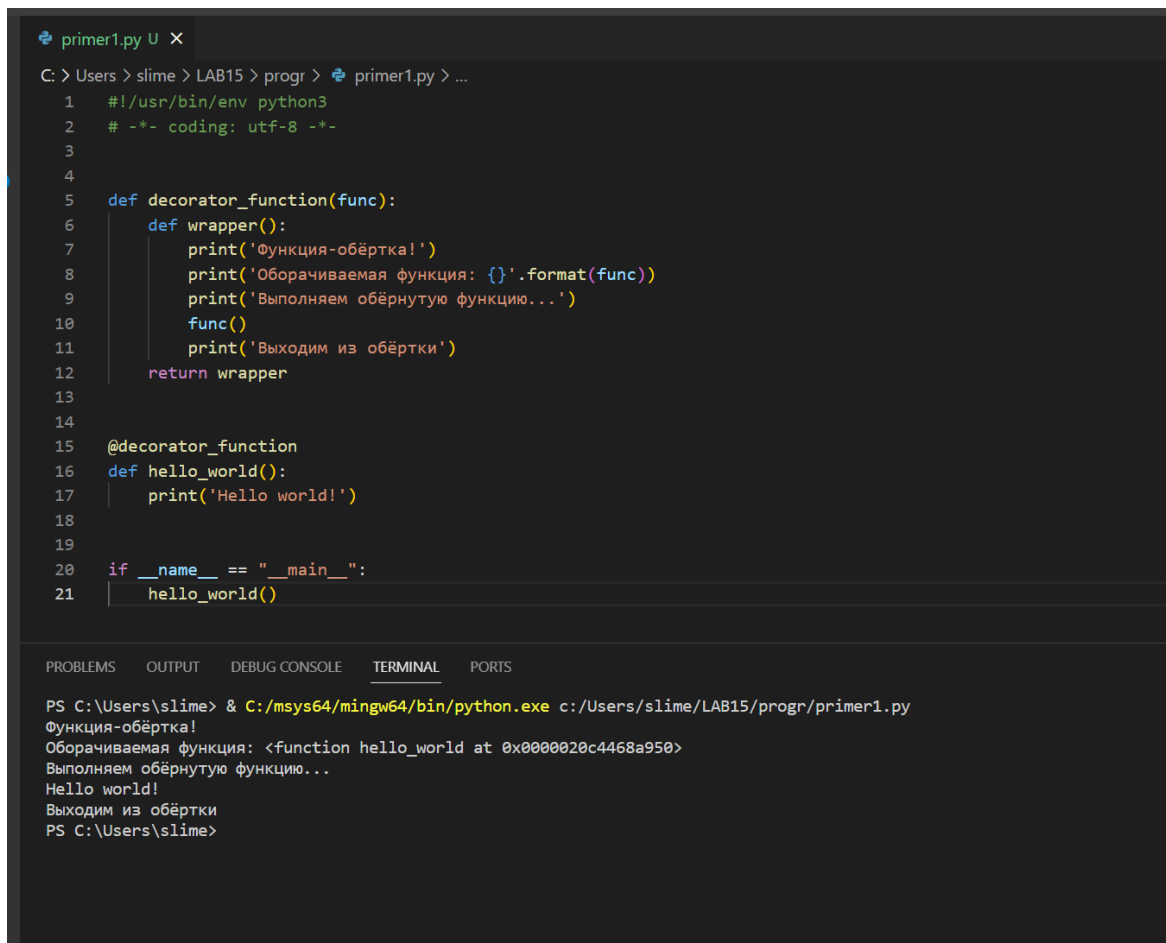
Ставрополь, 2023 г.

Тема: Декоратор функций в языке Python

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

Ход работы

1. Создал общедоступный репозиторий на GitHub, в котором использована лицензия MIT и язык программирования Python. Выполнил клонирование созданного репозитория.
2. Дополнил файл .gitignore необходимыми правилами.
3. Организовал созданный репозиторий в соответствие с моделью ветвления git-flow.
4. Проработал пример лабораторной работы. Создал для него отдельный модуль языка Python. Привел в отчете скриншоты результата выполнения программы примера при различных исходных данных, вводимых с клавиатуры.



```
primer1.py U X
C: > Users > slime > LAB15 > progr > primer1.py > ...
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4
5  def decorator_function(func):
6      def wrapper():
7          print('Функция-обёртка!')
8          print('Оборачиваемая функция: {}'.format(func))
9          print('Выполняем обёрнутую функцию...')
10         func()
11         print('Выходим из обёртки')
12     return wrapper
13
14
15 @decorator_function
16 def hello_world():
17     print('Hello world!')
18
19
20 if __name__ == "__main__":
21     hello_world()
```

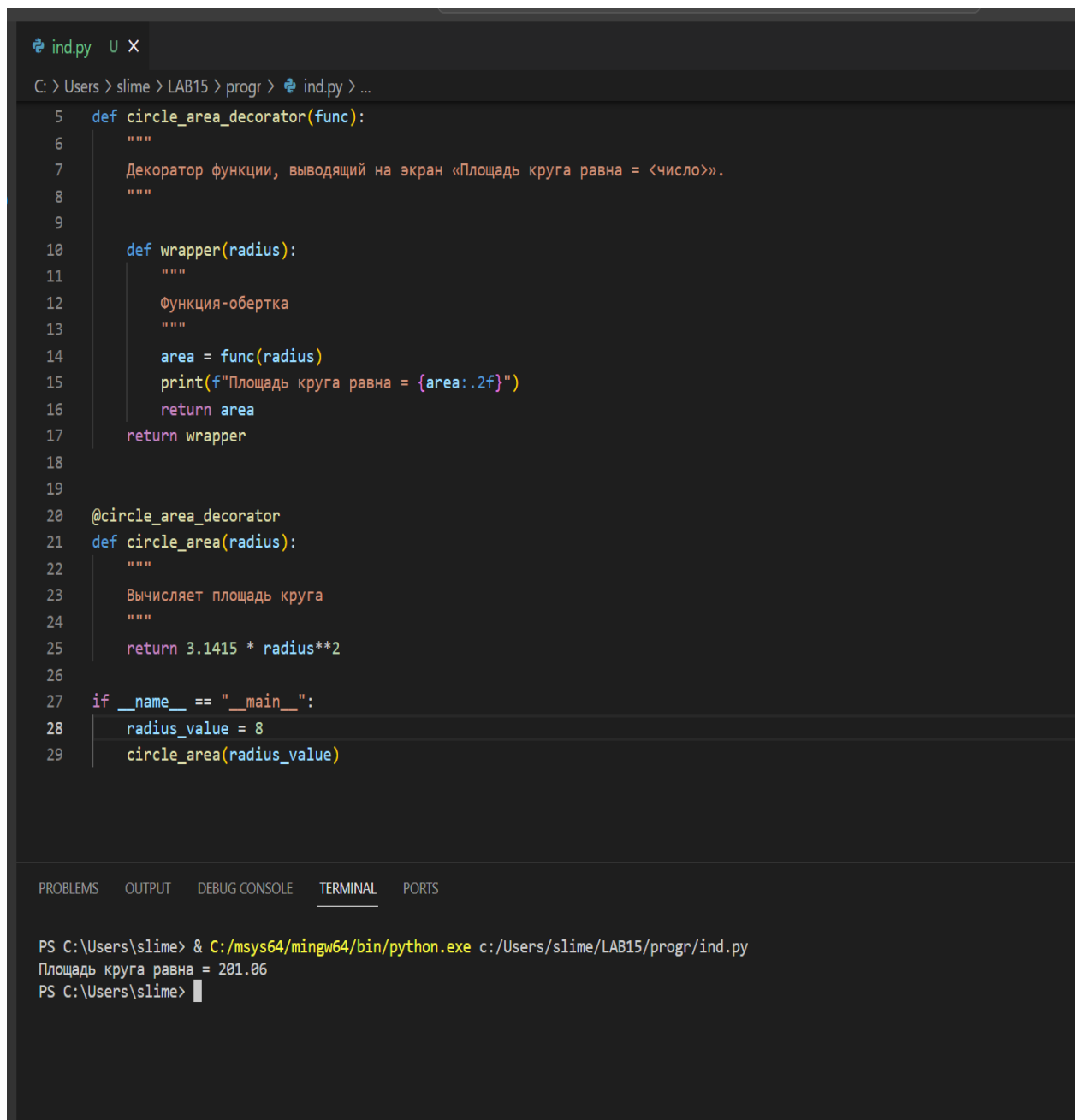
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\slime> & C:/msys64/mingw64/bin/python.exe c:/Users/slime/LAB15/progr/primer1.py
Функция-обёртка!
Оборачиваемая функция: <function hello_world at 0x0000020c4468a950>
Выполняем обёрнутую функцию...
Hello world!
Выходим из обёртки
PS C:\Users\slime>
```

Рисунок 1. Результат работы программы из примера 1

5. Выполнил индивидуальное задание, согласно варианту 8. Привёл в отчете скриншот работы программы.

Задание. Объявите функцию, которая вычисляет площадь круга и возвращает вычисленное значение. В качестве аргумента ей передается значение радиуса. Определите декоратор для этой функции, который выводит на экран сообщение: «Площадь круга равна = <число>». В строке выведите числовое значение с точностью до сотых. Примените декоратор к функции и вызовите декорированную функцию.



```
ind.py U X
C: > Users > slime > LAB15 > progr > ind.py > ...

5 def circle_area_decorator(func):
6     """
7     Декоратор функции, выводящий на экран «Площадь круга равна = <число>».
8     """
9
10    def wrapper(radius):
11        """
12        Функция-обертка
13        """
14        area = func(radius)
15        print(f"Площадь круга равна = {area:.2f}")
16        return area
17    return wrapper
18
19
20 @circle_area_decorator
21 def circle_area(radius):
22     """
23     Вычисляет площадь круга
24     """
25     return 3.1415 * radius**2
26
27 if __name__ == "__main__":
28     radius_value = 8
29     circle_area(radius_value)

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\slime> & C:/msys64/mingw64/bin/python.exe c:/Users/slime/LAB15/progr/ind.py
Площадь круга равна = 201.06
PS C:\Users\slime>
```

Рисунок 2. Результат работы программы из индивидуального задания

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?

Функции являются объектами первого класса, потому что они могут быть присвоены переменной, переданы в качестве аргумента другой функции, возвращены из функции и сохранены в структурах данных.

3. Каково назначение функций высших порядков?

Функции высших порядков могут принимать другие функции в качестве аргументов или возвращать их в качестве результатов. Они позволяют абстрагировать действия и оперировать функциями, как данными.

4. Как работают декораторы?

Декораторы работают путем обертывания другой функции, позволяя добавлять новое поведение к этой функции без изменения ее кода. Декораторы принимают функцию в качестве аргумента, возвращают другую функцию и обычно используются с символом `@`.

5. Какова структура декоратора функций?

Структура декоратора функций обычно выглядит следующим образом:

```
def decorator_function(original_function):  
    def wrapper_function(*args, **kwargs):  
        # Добавление нового поведения  
        return original_function(*args, **kwargs)  
    return wrapper_function
```

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

Параметры могут быть переданы декоратору, а не декорируемой функции, путем добавления еще одного уровня вложенной функции в

декораторе. Этот уровень может принимать параметры и передавать их в обернутую функцию.

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