

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:
Горбунов Данила Евгеньевич
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 _____

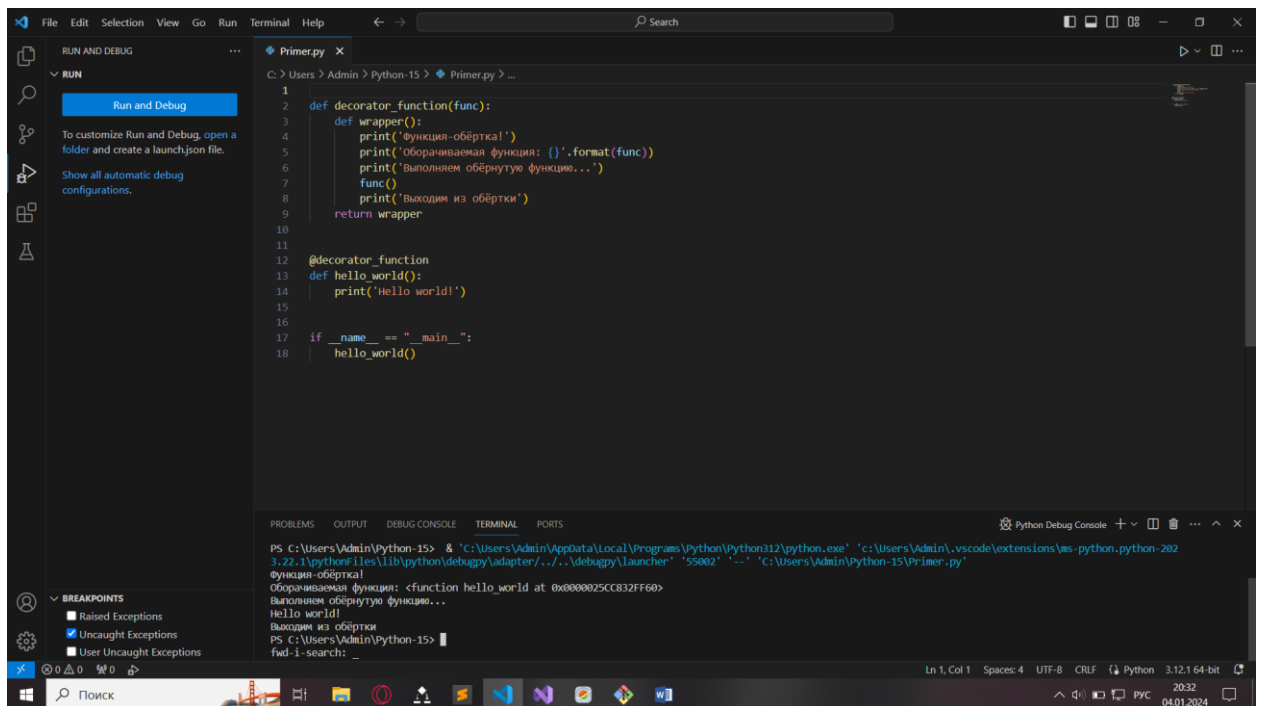
Ставрополь, 2023 г.

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.



The screenshot shows a Python IDE with a file named 'Primer.py'. The code defines a decorator function 'decorator_function' that takes a function 'func' and returns a wrapper. The wrapper prints 'Функция-обёртка!', then prints the function name and its address, then calls the function, and finally prints 'Выходим из обёртки!'. The decorator is applied to a function 'hello_world' which prints 'hello world!'. The main block calls 'hello_world()'. The output console shows the execution results: 'Функция-обёртка!', 'hello world!', and 'Выходим из обёртки!'. The terminal shows the command 'python primer.py' and the output 'hello world!'. The status bar at the bottom indicates 'Ln 1, Col 1', 'Spaces: 4', 'UTF-8', 'CRLF', 'Python', '3.12.1 64-bit', and the date '2032 04.01.2024'.

```
1 def decorator_function(func):
2     def wrapper():
3         print('Функция-обёртка!')
4         print('Оборачиваемая функция: {}'.format(func))
5         print('Выполняем обернутую функцию...')
6         func()
7         print('Выходим из обёртки!')
8     return wrapper
9
10
11
12 @decorator_function
13 def hello_world():
14     print('hello world!')
15
16
17 if __name__ == '__main__':
18     hello_world()
```

Python Debug Console

```
PS C:\Users\Admin\Python-15> & 'C:\Users\Admin\AppData\Local\Programs\Python\Python312\python.exe' 'C:\Users\Admin\.vscode\extensions\ms-python.python-2023.22.1\pythonfiles\lib\python\debugpy\adapter\..\..\debugpy\launcher' '55802' '-.' 'C:\Users\Admin\Python-15\Primer.py'
Функция-обёртка!
Оборачиваемая функция: <function hello_world at 0x0000025CC832FF60>
Выполняем обернутую функцию...
hello world!
Выходим из обёртки
PS C:\Users\Admin\Python-15>
```

Рисунок 1. Результат работы программы из примера 1

5. Выполнил индивидуальное задание. Привёл в отчете скриншот работы программы.

Объявите функцию, которая вычисляет периметр многоугольника и

возвращает вычисленное значение. Длины сторон многоугольника передаются в виде коллекции (списка или кортежа). Определите декоратор для этой функции, который выводит на экран сообщение: «Периметр фигуры равен = <число>». Примените декоратор к функции и вызовите декорированную функцию.

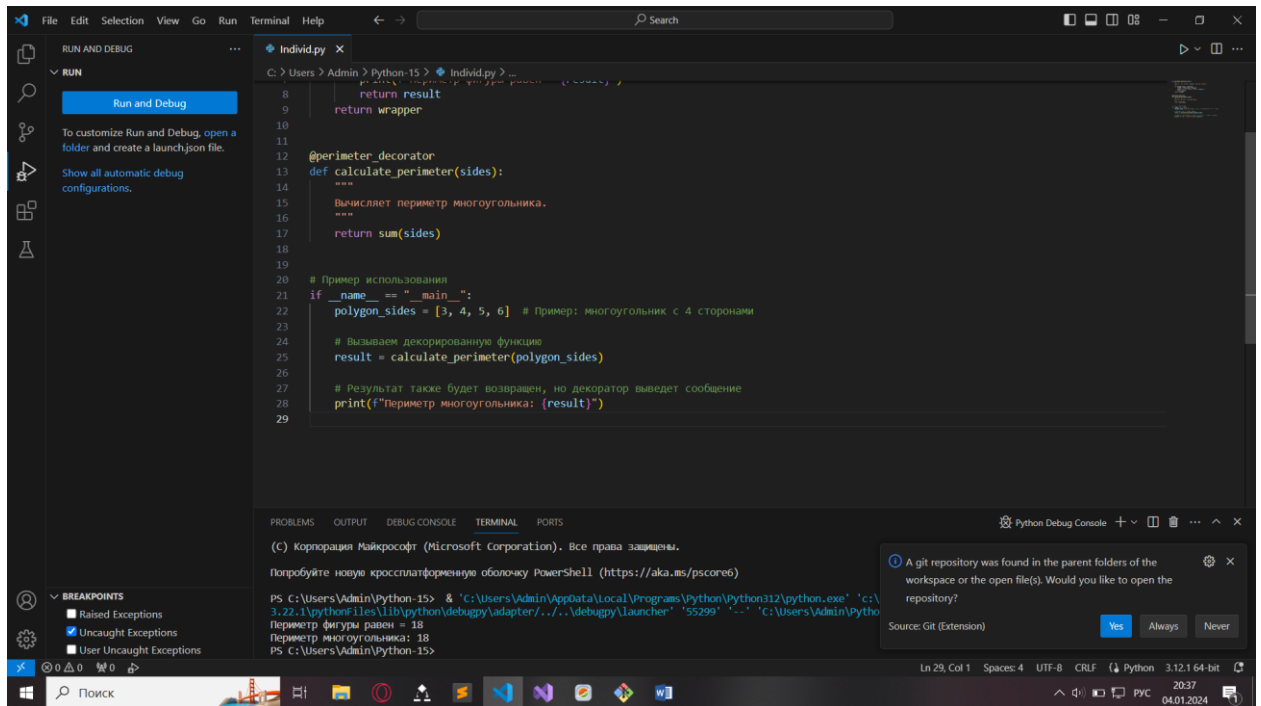


Рисунок 2. Результат работы программы из индивидуального задания

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