

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ
ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего
образования
«СЕВЕРОКАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

**Кафедра
инфокоммуникаций
Институт цифрового
развития**

**ОТЧЁТ
по лабораторной работе №2.12**

Дисциплина: «Программирование на
Python» Тема:

«Декораторы функций в языке Python»

Выполнил: студент 2 курса
группы ИВТ-б-о-21-1
Богдашов Артём
Владимирович

Ставрополь 2022

Выполнение работы:

1. Создал репозиторий в GitHub «геп Laba2.12» в который добавил .gitignore, который дополнил правила для работы с IDE PyCharm с ЯП Python, выбрал лицензию MIT, клонировал его на лок. сервер и организовал в соответствии смоделью ветвления git-flow.

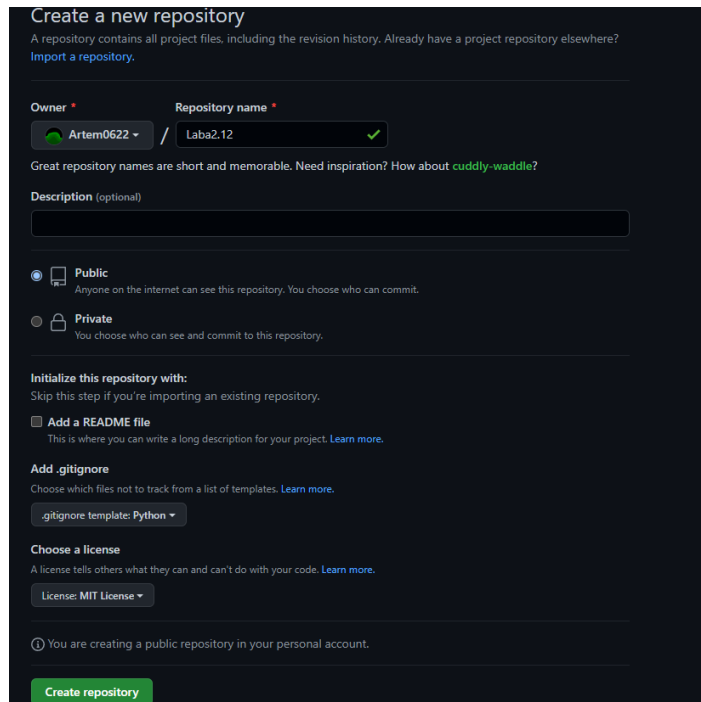


Рисунок 1.1 Создание репозитория

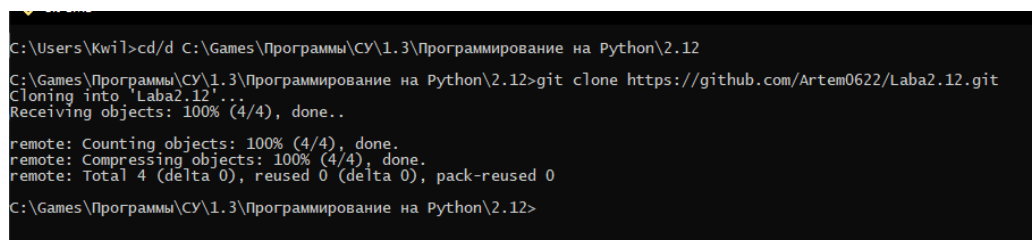


Рисунок 1.2 Клонирование репозитория

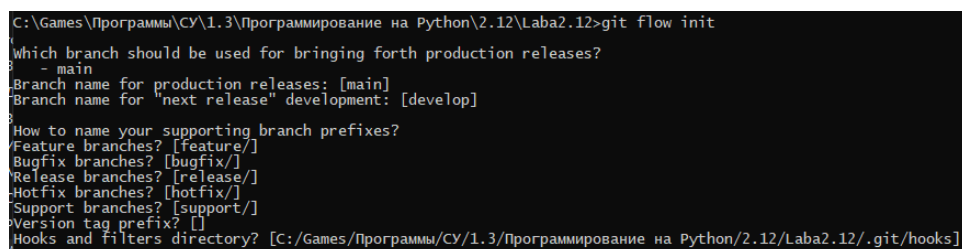


Рисунок 1.3 Организация репозитория в соответствии с моделью ветвления git-flow

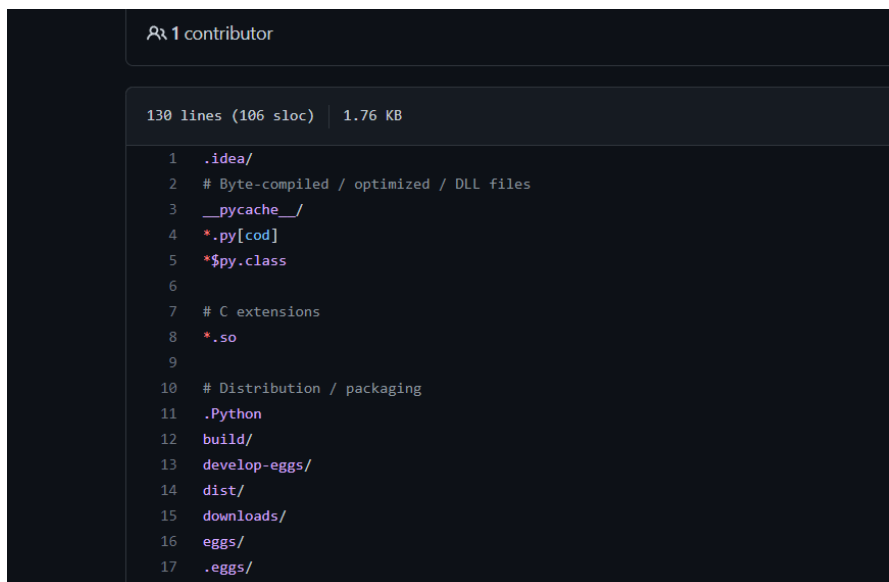


Рисунок 1.4 Изменение .gitignore

3.(3 вариант). Выполнил индивидуальное задание.

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.

```
1 def decor(func):
2     def wrapper(*args):
3         a1, a2 = func(*args)
4         return dict(zip(a1, a2))
5
6     return wrapper
7
8
9 def make_two_list(s1, s2):
10     return s1.split(), s2.split()
11
12
13 s1 = 'Вася Коля Маша Аня Таня'
14 s2 = 'мальчик мальчик девочка девочка девочка'
15
16 print(make_two_list(s1, s2))
17
```

Run: ind

C:\Users\Kwll\PycharmProjects\Lab2.10\venv\Scripts\python.exe "C:\Games\Программы\СУ\1.3\Программирование на Python\2.12\ind.py"

(['Вася', 'Коля', 'Маша', 'Аня', 'Таня'], ['мальчик', 'мальчик', 'девочка', 'девочка', 'девочка'])

Process finished with exit code 0

Рисунок 2.1 Вывод программы индивидуального задания

```
Git CMD
Version tag prefix? []
Hooks and filters directory? [C:/Games/Программы/СУ/1.3/Программирование на Python/2.12/Laba2.12/.git/hooks] git add .
C:\Games\Программы\СУ\1.3\Программирование на Python\2.12\Laba2.12>git commit -m "programs"
On branch develop
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    ind.py
    primer/

nothing added to commit but untracked files present (use "git add" to track)
C:\Games\Программы\СУ\1.3\Программирование на Python\2.12\Laba2.12>git add .
C:\Games\Программы\СУ\1.3\Программирование на Python\2.12\Laba2.12>git commit -m "save"
[develop 35f4e2d] save
5 files changed, 98 insertions(+)
create mode 100644 ind.py
create mode 100644 primer/primer.py
create mode 100644 primer/primer2.py
create mode 100644 primer/primer3.py
create mode 100644 primer/primer4.py
```

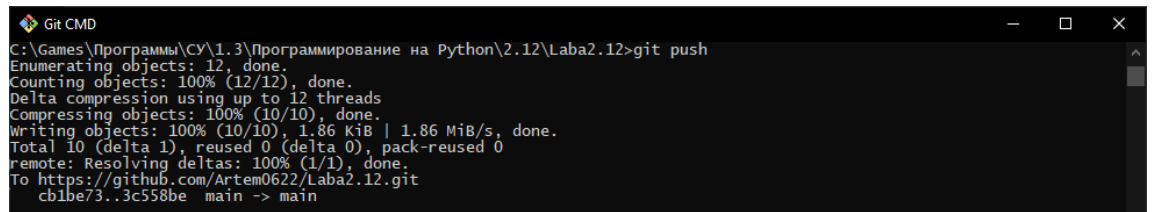
Рисунок 3.1 Коммит изменений

```
C:\Games\Программы\СУ\1.3\Программирование на Python\2.12\Laba2.12>git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
```

Рисунок 3.2 Переход на ветку main

```
C:\Games\Программы\СУ\1.3\Программирование на Python\2.12\Laba2.12>git merge develop
Updating 6230ccc..35f4e2d
Fast-forward
 ind.py | 16 ++++++++
 primer/primer.py | 12 +++++
 primer/primer2.py | 21 ++++++++
 primer/primer3.py | 23 ++++++++
 primer/primer4.py | 26 ++++++++
 5 files changed, 98 insertions(+)
 create mode 100644 ind.py
 create mode 100644 primer/primer.py
 create mode 100644 primer/primer2.py
 create mode 100644 primer/primer3.py
 create mode 100644 primer/primer4.py
C:\Games\Программы\СУ\1.3\Программирование на Python\2.12\Laba2.12>
```

Рисунок 3.3 Слияние ветки main с develop



```
Git CMD
C:\Games\Программы\СУ\1.3\Программирование на Python\2.12\Laba2.12>git push
Enumerating objects: 12, done.
Counting objects: 100% (12/12), done.
Delta compression using up to 12 threads
Compressing objects: 100% (10/10), done.
Writing objects: 100% (10/10), 1.86 KiB | 1.86 MiB/s, done.
Total 10 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), done.
To https://github.com/Artem0622/Laba2.12.git
   cb1be73..3c558be  main -> main
```

Рисунок 3.4 Пуш изменений

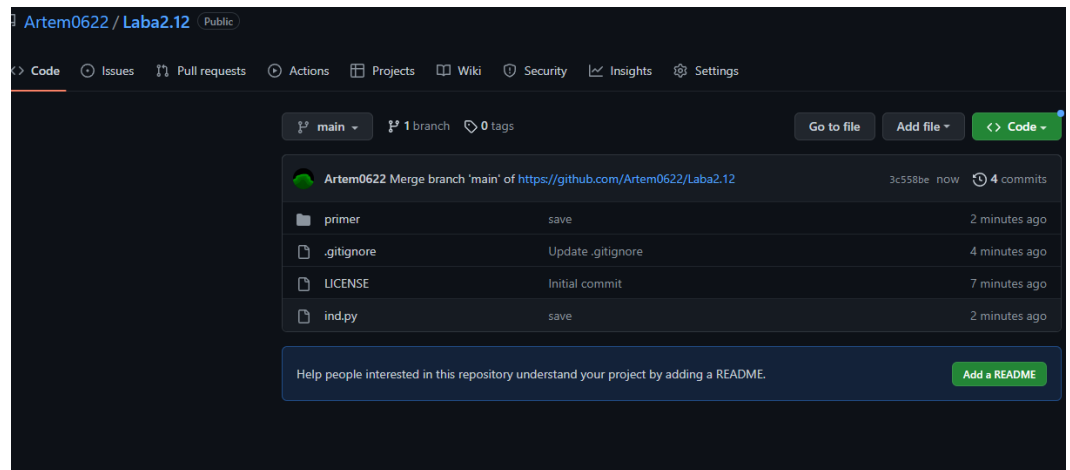


Рисунок 3.5 Изменение на уд сервере

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.