Федеральное государственное автономное

образовательное учреждение

высшего образования

«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий

институт

Кафедра «Информатика»

кафедра

**ОТЧЕТ О ПРАКТИЧЕСКОЙ** **РАБОТЕ №5**

Документирование программного кода

Тема

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Преподаватель |  |  |  |  |  | П.В.Пересунько |
|  |  |  |  | подпись, дата |  | инициалы, фамилия |
| Студент |  | КИ19-16/2б |  |  |  | С.Е.Шпаков |
|  |  | номер группы, зачетной книжки |  | подпись, дата |  | инициалы, фамилия |

Красноярск 2019

# СОДЕРЖАНИЕ

[1 Цель и задачи 3](#_Toc27678199)

[1.1 Цель 3](#_Toc27678200)

[1.2 Задачи 3](#_Toc27678201)

[2 Описание варианта задания 3](#_Toc27678202)

[2.1 Основное задание 3](#_Toc27678203)

[2.2 Дополнительное задание 4](#_Toc27678204)

[3 Ход выполнения 4](#_Toc27678205)

[4 Выводы 11](#_Toc27678206)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 12](#_Toc27678207)

# 1 Цель и задачи

## 1.1 Цель

Изучить принципы составления программной документации и оформления README.md файла на удаленном репозитории. Научиться писать документацию на Doxygen.

## 1.2 Задачи

Для выполнения практической работы необходимо выполнить следующие задачи:

1. написать документацию для Doxygen для **всех** функций в своей предыдущей практической работе;
2. документация для модельных тестов, где перед функцией теста указывается какой тестовый случай рассматривается;
3. сгенерировать документацию кода с помощью Doxygen и получить html страничку документации;
4. залить документацию на удаленный репозиторий;
5. оформить файл README.md на удаленном репозитории gitlab;
6. оформить отчет по практической работе ☺.

# 2 Описание варианта задания

## 2.1 Основное задание

Вариант №23. Имеется алфавит *А*, состоящий только из восьми букв – «*О*», «*Е*», «*W*», «*M*», «*C*», «*A*», «*U*», «*Q*». Для каждого символа исходного алфавита *А* в соответствие поставлена комбинация символов из кодового алфавита *B*:

1. «*O*» - кодируется комбинацией «\_\_\_»;
2. «*E*» - кодируется комбинацией «\_\_.»;
3. «*W*» - кодируется комбинацией «\_.\_»;
4. «*M*» - кодируется комбинацией «\_..»;
5. «*C*» - кодируется комбинацией «.\_\_»;
6. «*A*» - кодируется комбинацией «.\_.»;
7. «*U*» - кодируется комбинацией «..\_»;
8. «*Q*» - кодируется комбинацией «…»;

Необходимо раскодировать строку, состоящую из букв кодового алфавита *B*.

## 2.2 Дополнительное задание

Написать документацию для каждой функции в программном коде (указание аргументов функции, возвращаемое значение, возможные исключения, примеры запуска). Создать файл README.md в репозитории.

# 3 Ход выполнения

В самую первую очередь необходимо добавить комментарии (документацию) к функциям в программном коде. В итоге долгого и муторного подбирания слов дело приходит к следующему виде (листинг 1).

Листинг 1 – Реализованная программа с комментариями

## @package meow

# @author Xassie

# @date 18 Dec 2019

# @brief Позовите на помощь.

import argparse

## Кодовый алфавит в виде словаря-константы.

ALPHABET = {

'\_\_\_': 'O',

'\_\_.': 'E',

'\_.\_': 'W',

'\_..': 'M',

'.\_\_': 'C',

'.\_.': 'A',

'..\_': 'U',

'...': 'Q',

}

def parse():

"""! Принимает аргумент через командную строку.

Принимает драгоценное сообщение пользователя, которое его

величество желает декодировать. Сама по себе функция ничего

не принимает.

Args:

None

Returns:

R.n (str): Ввод пользователя при запуске через командную строку.

Raises:

None

Examples:

>>> python meow.py 'something'

'something'

>>> python meow.py '\_\_\_\_.\_\_\_.\_\_.'

'\_\_\_\_.\_\_\_.\_\_.'

"""

parser = argparse.ArgumentParser(description="Give a line to decode")

parser.add\_argument('n', type=str, help='Coded message')

R = parser.parse\_args()

return R.n

def decode(message):

"""! Декодирует сообщение на основе известного кодового алфавита.

Принимает закодированное сообщение и декодирует его на

основе заранее известных соответствий алфавитов.

Args:

message (str): Сообщение для декодирования.

dc (str): Поочереди собирает элементы message.

res (str): Собирает декодированные символы из dc.

Returns:

res: Результат расшифровки. Если невозможно декодировать, то None.

Raises:

TypeError

Examples:

>>> decode('\_\_\_\_.\_\_\_\_')

'OWO'

>>> decode('\_\_\_..')

None

>>> decode('SAUCE')

None

>>> decode(2020)

Traceback (most recent call last):

...

TypeError: 'int' object is not iterable

"""

dc = ''

res = ''

for i in message:

dc += i

if dc in ALPHABET:

res += ALPHABET[dc]

dc = ''

if not dc:

return res

def mcheck(message):

"""! Проверка на отсутствие посторонних символов.

Проверяет есть ли в поданном сообщении символы,

отсутствущие в кодовом алфавите. Принимает сообщение.

Если все символы верны, то возвращает 1. В противном

случае None.

Args (str):

message: Пользовательское сообщение.

Returns:

Около-булевое значение как результат проверки.

Raises:

TypeError

Examples:

>>> mcheck('.\_\_..\_\_..')

1

>>> mcheck('\_')

1

>>> mcheck('Friendship is magic!')

None

>>> mcheck(2020)

Traceback (most recent call last):

...

TypeError: 'int' object is not iterable

"""

if set(message).union(set(['\_', '.'])) == {'.', '\_'}:

return 1

def main():

"""! Основная логика программы.

Приветствует Мир-тян. Регулирует последовательность выполнения функций.

Делает всю грязную работу. Выводит пользователю в консоль результат.

Args:

msg (str): Зашифрованное сообщение.

result (str): Результат расшифровки

Returns:

Сама по себе функция ничего не возвращает.

Raises:

None

Examples:

>>> main()

None

"""

print('Hello world-chan!')

msg = parse()

if mcheck(msg):

result = decode(msg)

if result:

print(result)

else:

print('Impossible to decode')

else:

print('Please enter valid message')

if \_\_name\_\_ == '\_\_main\_\_':

main()

Теперь всё переходит к гвоздю программы, а именно к Doxygen. Опустим установку, перейдем сразу к действию. После запуска необходимо произвести первоначальную настройку, а именно вписать директорию с кодом и директорию для документации (рисунок 1).

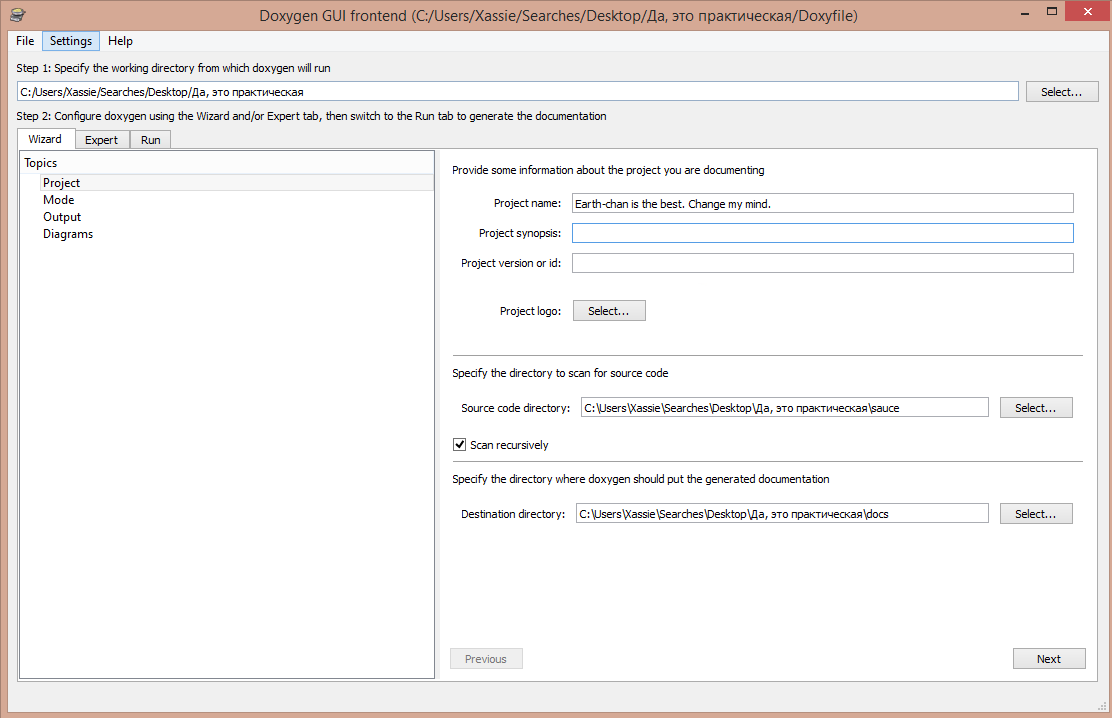


Рисунок 1 – Первоначальная настройка Doxygen

Дальше сцена переходит на вкладку Expert и там мы ставим язык документации и включаем самые красивые галочки (рисунок 2).

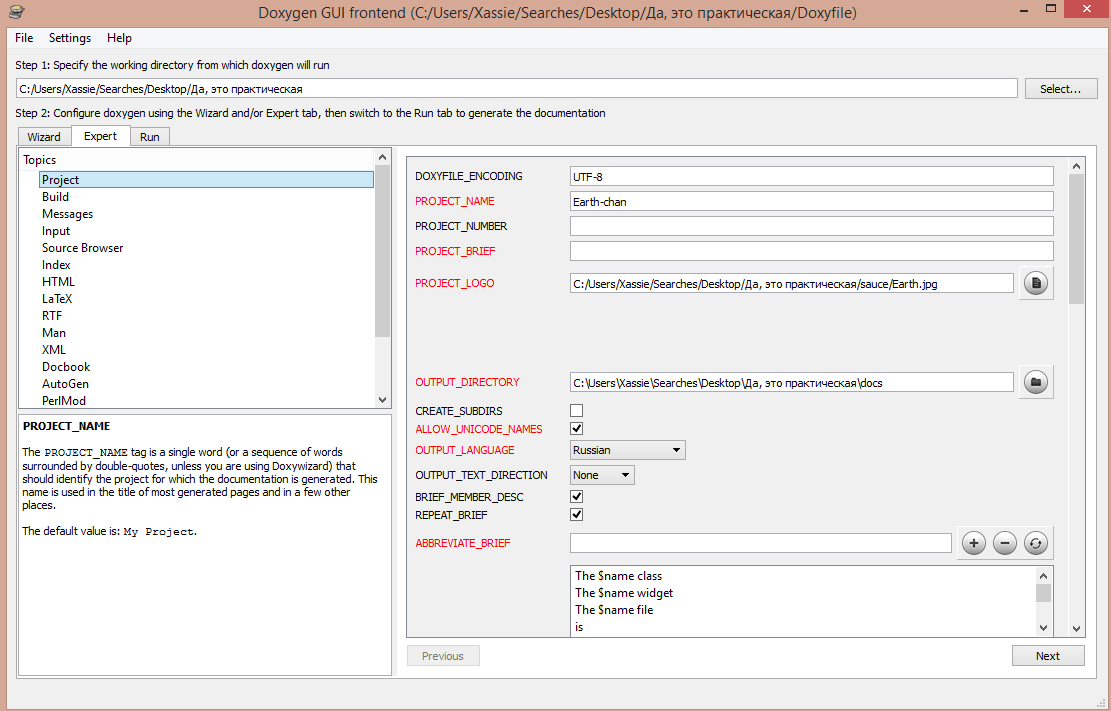


Рисунок 2 – Расширенные настройки Doxygen

Следующий шаг самый важный. Необходимо потанцевать с бубном, помолиться богу программного кода и самое главное – поблагодарить Землю-чан (рисунок 3).

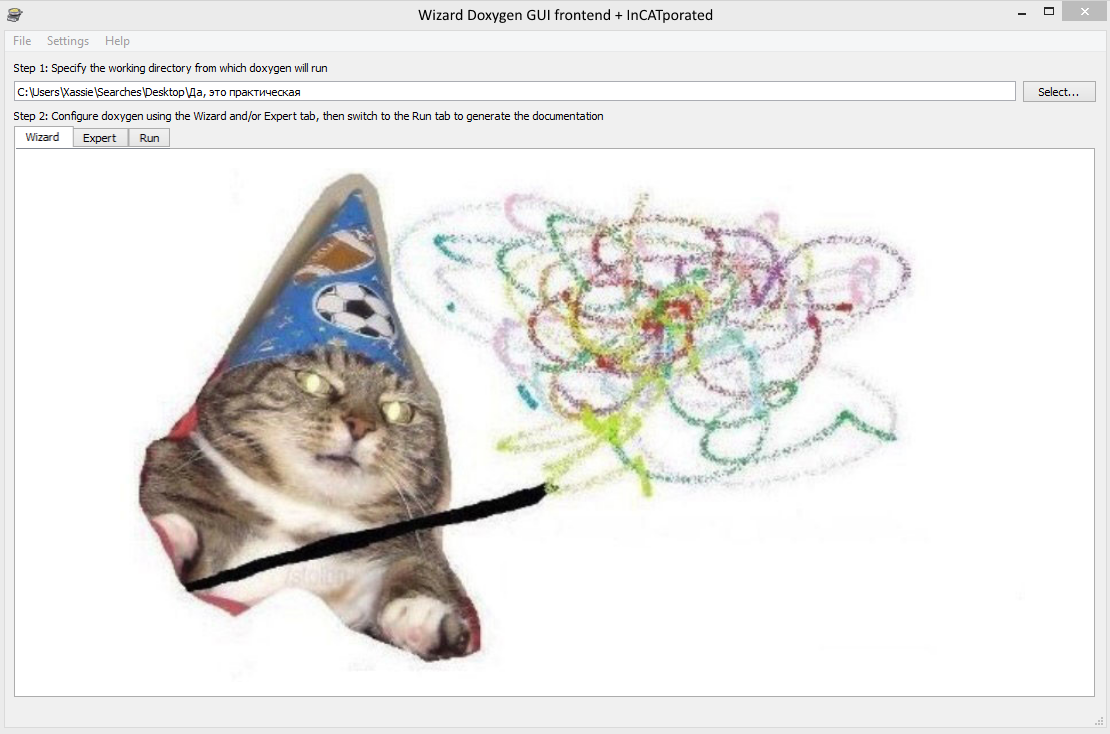


Рисунок 3 – Особые действия, чтобы Doxygen работал с Python

После этого мы можем наконец-то нажать кнопку запуска “Run Doxygen” и он нам создаст документацию в выбранной директории (рисунок 4). По умолчанию он создаст html и Latex, но в настройках можно включить и другие форматы.

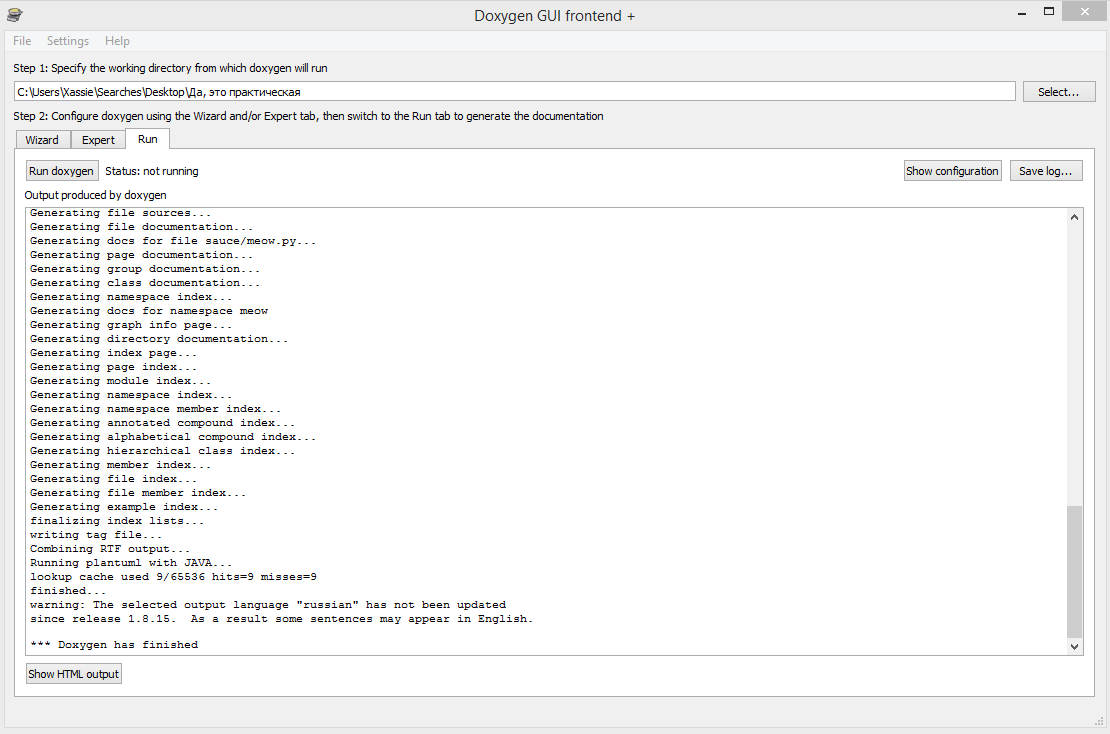


Рисунок 4 – Генерация документации в Doxygen

Всё готово! С помощью кнопки Show HTML output или открыв файл index.html в директории мы можем просмотреть получившуюся документацию. Для лучшего восприятия можно внедрить функции Doxygen в код, тогда всё будет выглядеть примерно так (рисунок 5).

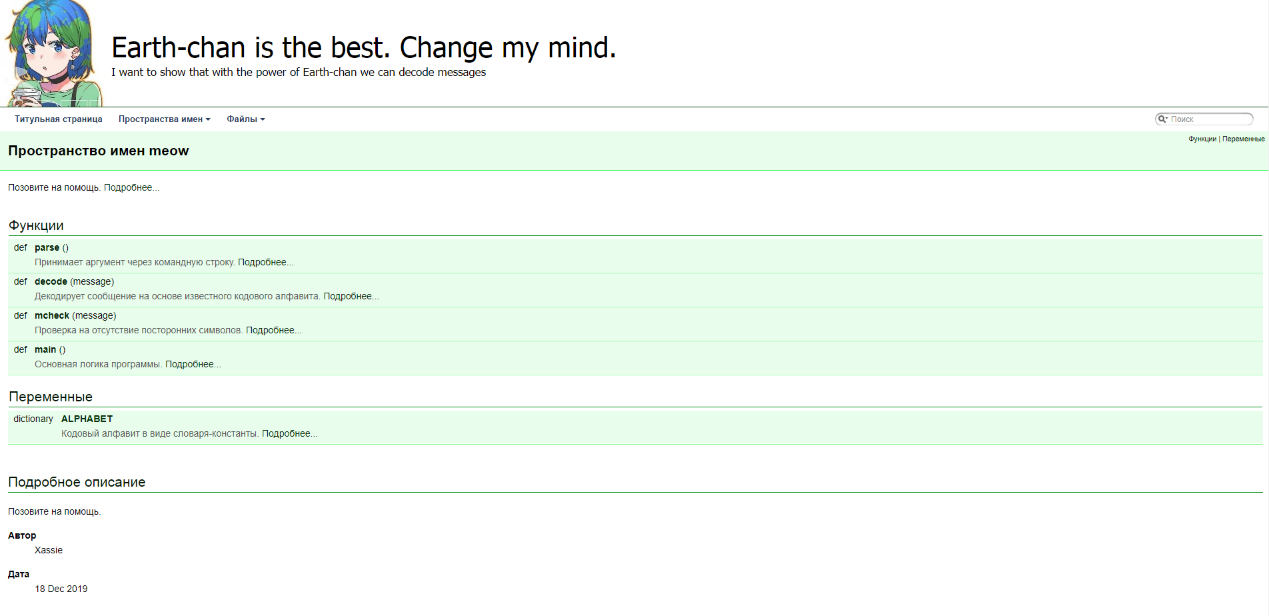


Рисунок 5 – Готовая документация

И в самую последнюю очередь необходимо написать описание проекта – файл README.md в удаленном репозитории. Делается это непосредственно на сайте GitLab (хотя это совсем не обязательно) и после длительного описания различной информации получается полноценный readme (рисунок 6).

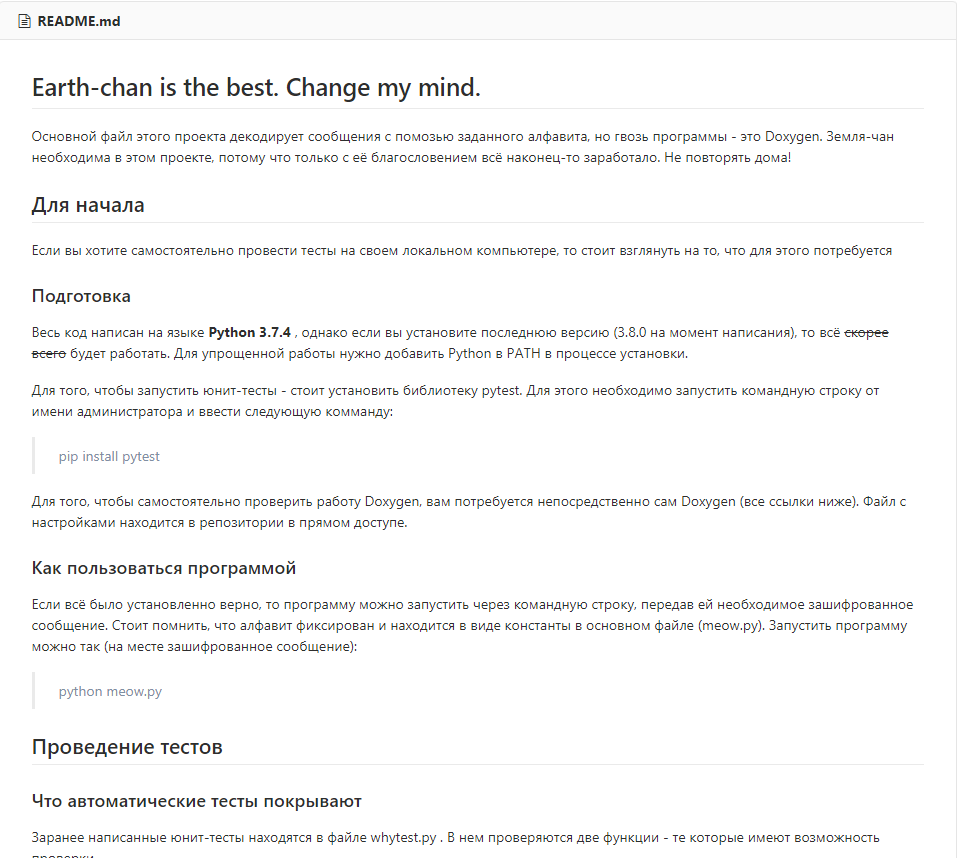


Рисунок 6 – README.md в удаленном репозитории

# 4 Выводы

В результате работы составлена документация для программы на русском языке и скомпонована с помощью Doxygen. В придачу создан и реализован файл README.md для всего удаленного репозитория GitLab.

# СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. СТО 4.2-07-2014 Система менеджмента качества Общие требования к построению, изложению и оформлению документов учебной деятельности. – Взамен СТО 4.2–07–2012 ; введ. 09.01.2014, - Красноярск : ИПК СФУ, 2014 – 60 с.