

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ
ФЕДЕРАЦИИ**

**Федеральное государственное автономное
образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

**Кафедра инфокоммуникаций
«Основы цифровой обработки изображений в OpenCv»**

**Отчет по лабораторной работе № 2.22(9)
по дисциплине «Основы программной инженерии»**

Выполнил студент группы ПИЖ-б-о-21-1

Халимендик Я. Д. « » 2023г.

Подпись студента _____

Работа защищена « » _____ 20__ г.

Проверил Воронкин Р.А. _____
(подпись)

Ставрополь 2023

Цель работы: приобретение навыков написания автоматизированных тестов на языке программирования Python версии 3.x.

Ход работы:

1. Изучить теоретический материал работы.
2. Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия IT и язык программирования Python.

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner * Yana-Kh / **Repository name *** Lab-2.21-OPJ

⚠ The repository Lab-2.21-OPJ already exists on this account.

Great repository names are short and memorable. Need inspiration? How about [automatic-guide?](#)

Description (optional)

☒ **Public**
Anyone on the internet can see this repository. You choose who can commit.

☐ **Private**
You choose who can see and commit to this repository.

Initialize this repository with:

☐ **Add a README file**
This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore

.gitignore template: **None**

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license

License: **None**

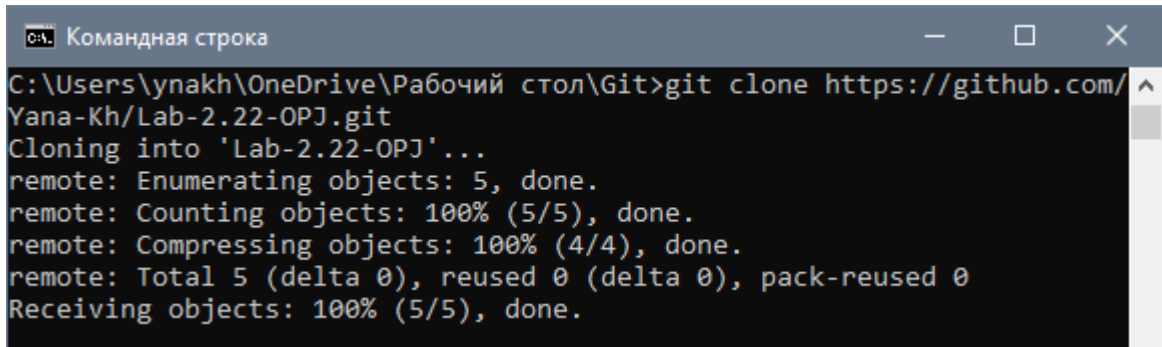
A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

Create repository

ⓘ You are creating a public repository in your personal account.

Рисунок 1 – Создание репозитория

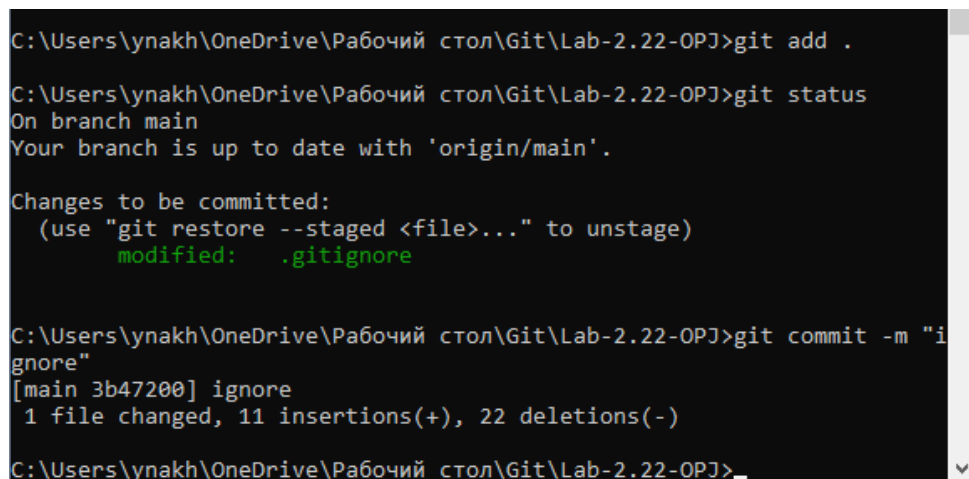
3. Выполните клонирование созданного репозитория.



```
Командная строка
C:\Users\ynakh\OneDrive\Рабочий стол\Git>git clone https://github.com/Yana-Kh/Lab-2.22-OPJ.git
Cloning into 'Lab-2.22-OPJ'...
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (5/5), done.
```

Рисунок 2 – Клонирование репозитория

4. Дополните файл .gitignore необходимыми правилами для работы с IDE PyCharm.



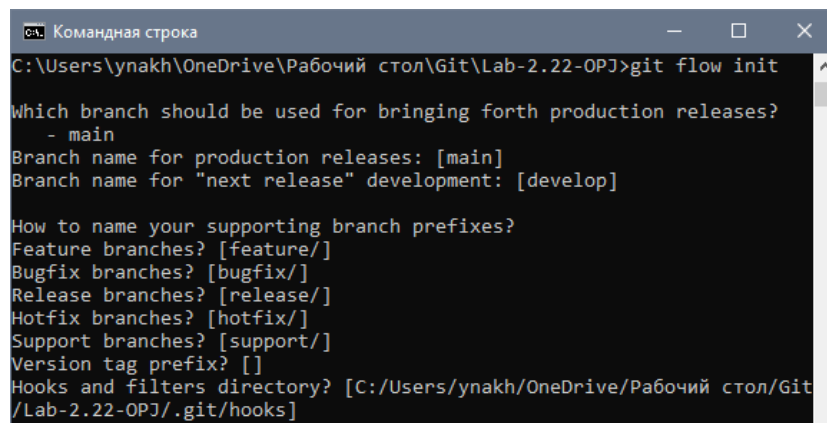
```
C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-2.22-OPJ>git add .
C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-2.22-OPJ>git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   .gitignore

C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-2.22-OPJ>git commit -m "ignore"
[main 3b47200] ignore
 1 file changed, 11 insertions(+), 22 deletions(-)
C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-2.22-OPJ>
```

Рисунок 3 – Дополнение файла .gitignore

5. Организуйте свой репозиторий в соответствии с моделью ветвления git-flow.



```
Командная строка
C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-2.22-OPJ>git flow init
Which branch should be used for bringing forth production releases?
- main
Branch name for production releases: [main]
Branch name for "next release" development: [develop]

How to name your supporting branch prefixes?
Feature branches? [feature/]
Bugfix branches? [bugfix/]
Release branches? [release/]
Hotfix branches? [hotfix/]
Support branches? [support/]
Version tag prefix? []
Hooks and filters directory? [C:/Users/ynakh/OneDrive/Рабочий стол/Git/Lab-2.22-OPJ/.git/hooks]
```

Рисунок 4 – Организация репозитория в соответствии с моделью git-flow

6. Создайте проект PyCharm в папке репозитория.

Рисунок 5 – Создание проекта PyCharm

7. Выполните индивидуальное задание. Приведите в отчете скриншоты работы программы решения индивидуального задания.

Задание: Для индивидуального задания лабораторной работы 2.21 добавьте тесты с использованием модуля unittest, проверяющие операции по работе с базой данных.

Код:

tests_people.py

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

import unittest
import sqlite3
from pathlib import Path
from id_21 import create_db, add_human, select_all, find_human

class HumanTests(unittest.TestCase):
    # Метод действует на уровне класса, т.е. выполняется
    # перед запуском тестов класса.
    @classmethod
    def setUpClass(cls):
        """Set up for class"""

        print("setUpClass")
        print("=====")

    # Запускается после выполнения всех методов класса
    @classmethod
    def tearDownClass(cls):
        """Tear down for class"""
        print("=====")
        print("tearDownClass")

    # Метод вызывается перед запуском теста. Как правило,
    # используется для подготовки окружения для теста.
    def setUp(self):
        """Set up for test"""
        print(f"\nSet up for [{self.shortDescription()}]")

    # Метод вызывается после завершения работы теста.
    # Используется для "приборки" за тестом.
    def tearDown(self):
        """Tear down for test"""
        print(f"Tear down for [{self.shortDescription()}]")

    def test_create_db(self):
        """Checking the database creation."""
```

```

        database_path = "test.db"
        if Path(database_path).exists():
            Path(database_path).unlink()

        create_db(database_path)
        self.assertTrue(Path(database_path).is_file())
        Path(database_path).unlink()

    def test_add_human(self):
        """ Checking the addition. """
        database_path = "test.db"
        create_db(database_path)
        add_human(database_path, "Халимендик Яна", 9188871234, "05.05.2003")
        conn = sqlite3.connect(database_path)
        cursor = conn.cursor()
        cursor.execute(
            """
            SELECT * FROM people
            """
        )
        row = cursor.fetchone()
        self.assertEqual(row, (1, "Халимендик Яна", "05.05.2003"))
        conn.close()
        Path(database_path).unlink()

    def test_select_all(self):
        """ Checking the selection all """
        database_path = "test.db"
        create_db(database_path)
        add_human(database_path, "Иванов Иван", "10.09.2001", 9876754327)
        add_human(database_path, "Сидоров Сидр", "31.08.2017", 9871112233)

        comparison_output = [
            {"name": "Иванов Иван", "phone": '9876754327', "birthday":
"10.09.2001"},
            {"name": "Сидоров Сидр", "phone": '9871112233', "birthday":
"31.08.2017"},
        ]
        self.assertEqual(select_all(database_path), comparison_output)
        Path(database_path).unlink()

    def test_select_zz(self):
        """ Checking the selection by phone number """
        database_path = "test.db"
        create_db(database_path)
        add_human(database_path, "Иванов Иван", 9876754327, "10.09.2001")
        add_human(database_path, "Сидоров Сидр", 9871112233, "31.08.2017")
        comparison_output = [
            {"name": "Иванов Иван", "phone": 9876754327, "birthday":
"10.09.2001"},
        ]
        self.assertEqual(find_human(database_path, "Иванов"),
comparison_output)
        Path(database_path).unlink()

```

tests_ranner.py

```

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

import unittest
import tests_people

```

```

prodTestSuite = unittest.TestSuite()
prodTestSuite.addTest(unittest.makeSuite(tests_people.HumanTests))
print("count of tests: " + str(prodTestSuite.countTestCases()) + "\n")

runner = unittest.TextTestRunner(verbosity=2)
runner.run(prodTestSuite)

```

```

cmd. Командная строка
OK
C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-2.22-OPJ\py>tests_runner.py
count of tests: 4

setUpClass
=====
test_add_human (tests_people.HumanTests)
Checking the addition. ...
Set up for [Checking the addition.]
Tear down for [Checking the addition.]
ok
test_create_db (tests_people.HumanTests)
Checking the database creation. ...
Set up for [Checking the database creation.]
Tear down for [Checking the database creation.]
ok
test_select_all (tests_people.HumanTests)
Checking the selection all ...
Set up for [Checking the selection all]
Tear down for [Checking the selection all]
ok
test_select_zz (tests_people.HumanTests)
Checking the selection by phone number ...
Set up for [Checking the selection by phone number]
Tear down for [Checking the selection by phone number]
ok
=====
tearDownClass

-----
Ran 4 tests in 0.033s

```

Рисунок 6 – Результат работы программы

Вопросы для защиты работы:

1. Для чего используется автономное тестирование?

Автономное тестирование используется для автоматизации процесса проверки качества программного обеспечения. Вместо ручного выполнения тестовых сценариев автономное тестирование позволяет создавать и запускать тесты с использованием специальных инструментов или программных скриптов.

2. Какие фреймворки Python получили наибольшее распространение для решения задач автономного тестирования?

В мире Python существуют три framework'а, которые получили наибольшее распространение: unittest, nose, pytest.

3. Какие существуют основные структурные единицы модуля unittest?

Test fixture – обеспечивает подготовку окружения для выполнения тестов, а также организацию мероприятий по их корректному завершению (например очистка ресурсов). Подготовка окружения может включать в себя создание баз данных, запуск необходим серверов и т.п.

Test case – это элементарная единица тестирования, в рамках которой проверяется работа компонента тестируемой программы (метод, класс, поведение и т. п.). Для реализации этой сущности используется класс TestCase.

Test suite – это коллекция тестов, которая может в себя включать как отдельные test case'ы так и целые коллекции (т.е. можно создавать коллекции коллекций). Коллекции используются с целью объединения тестов для совместного запуска.

Test runner – это компонент, которые оркестрирует (координирует взаимодействие) запуск тестов и предоставляет пользователю результат их выполнения. Test runner может иметь графический интерфейс, текстовый интерфейс или возвращать какое-то заранее заданное значение, которое будет описывать результат прохождения тестов.

4. Какие существуют способы запуска тестов unittest?

Запуск тестов можно сделать как из командной строки, так и с помощью графического интерфейса пользователя (GUI)

5. Каково назначение класса TestCase?

Test case – это элементарная единица тестирования, в рамках которой проверяется работа компонента тестируемой программы (метод, класс, поведение и т. п.). Для реализации этой сущности используется класс TestCase.

6. Какие методы класса TestCase выполняются при запуске и завершении работы тестов?

`setUp()` – Метод вызывается перед запуском теста. Как правило, используется для подготовки окружения для теста.

`tearDown()` – Метод вызывается после завершения работы теста. Используется для “приборки” за тестом.

`setUpClass()` – Метод действует на уровне класса, т.е. выполняется перед запуском тестов класса. При этом синтаксис требует наличие декоратора `@classmethod`.

`tearDownClass()` – Запускается после выполнения всех методов класса, требует наличия декоратора `@classmethod`.

`skipTest(reason)` – Данный метод может быть использован для пропуска теста, если это необходимо.

7. Какие методы класса TestCase используются для проверки условий и генерации ошибок?

`assertEqual(a, b):`

Этот метод проверяет, равны ли значения `a` и `b`. Если значения не равны, то тест считается неудачным, и генерируется ошибка.

`assertNotEqual(a, b):`

Этот метод проверяет, не равны ли значения `a` и `b`. Если значения равны, то тест считается неудачным, и генерируется ошибка.

`assertTrue(x):`

Этот метод проверяет, является ли значение `x` истинным (равным `True`). Если значение `x` не является истинным, то тест считается неудачным, и генерируется ошибка.

`assertFalse(x):`

Этот метод проверяет, является ли значение `x` ложным (равным `False`). Если значение `x` не является ложным, то тест считается неудачным, и генерируется ошибка.

`assertIs(a, b):`

Этот метод проверяет, являются ли объекты `a` и `b` одним и тем же объектом (ссылаются ли они на одно и то же место в памяти). Если объекты не являются одним и тем же объектом, то тест считается неудачным, и генерируется ошибка.

`assertIsNot(a, b):`

Этот метод проверяет, не являются ли объекты `a` и `b` одним и тем же объектом. Если объекты являются одним и тем же объектом, то тест считается неудачным, и генерируется ошибка.

8. Какие методы класса `TestCase` позволяют собирать информацию о самом тесте?

`countTestCases()` – Возвращает количество тестов в объекте класса-наследника от `TestCase`.

`id()` – Возвращает строковый идентификатор теста. Как правило это полное имя метода, включающее имя модуля и имя класса.

`shortDescription()` – Возвращает описание теста, которое представляет собой первую строку `docstring`'а метода, если его нет, то возвращает `None`.

9. Каково назначение класса `TestSuite`? Как осуществляется загрузка тестов?

Класс `TestSuite` используется для объединения тестов в группы, которые могут включать в себя как отдельные тесты так и заранее созданные группы. Помимо этого, `TestSuite` предоставляет интерфейс, позволяющий `TestRunner`'у, запускать тесты.

10. Каково назначение класса `TestResult`?

Класс `TestResult` используется для сбора информации о результатах прохождения тестов.

11. Для чего может понадобиться пропуск отдельных тестов?

Пропуск отдельных тестов может быть полезным для временного отключения тестов, неподдерживаемых платформ или конфигураций, тестов в разработке или устаревших тестов.

12. Как выполняется безусловный и условных пропуск тестов? Как выполнить пропуск класса тестов?

Для пропуска теста воспользуемся декоратором, который пишется перед тестом: `@unittest.skip(reason)`

Для условного пропуска тестов применяются следующие декораторы: `@unittest.skipIf(condition, reason)` (Тест будет пропущен, если условие (condition) истинно) и `@unittest.skipUnless(condition, reason)` (Тест будет пропущен если, условие (condition) не истинно).

Для пропуска классов используется декоратор `@unittest.skip(reason)`

13. Самостоятельно изучить средства по поддержке тестов unittest в PyCharm. Приведите обобщенный алгоритм проведения тестирования с помощью PyCharm.

Алгоритм проведения тестирования с использованием PyCharm:

1. Настройка окружения тестирования.

Установите и настройте PyCharm для проекта, включая настройку интерпретатора Python, настройку путей к исходным файлам и тестовым файлам, а также настройку модуля unittest.

2. Создание тестовых файлов.

Создайте файлы с тестами в проекте. Обычно тестовые файлы содержат классы наследующие `unittest.TestCase` и содержащие тестовые методы.

3. Определение тестовых методов.

Определите тестовые методы внутри классов, используя методы `assert` для проверки ожидаемых результатов. Каждый тестовый метод должен начинаться с префикса "test".

4. Настройка конфигурации тестирования.

В PyCharm выберите конфигурацию для запуска тестов. Вы можете выбрать определенный тестовый файл, папку с тестами или использовать общую конфигурацию для запуска всех тестов в проекте.

5. Запуск тестов.

Запустите тесты, выбрав соответствующую конфигурацию тестирования. PyCharm выполнит все тестовые методы и выведет результаты выполнения в специальной панели или в консоли.

6. Анализ результатов тестирования.

7. Исправление проблем в коде или тестах.

8. Повторение итераций до достижения требуемого поведения программы.