**МИНИCTEPCTBO НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**Федеральное государственное автономное образовательное учреждение высшего образования**

**«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

**Кафедра инфокоммуникаций**

**«Разработка приложений с интерфейсом командной строки (CLI) в Python3»**

**Отчет по лабораторной работе № 2.17 по дисциплине «Основы программной инженерии»**

Выполнил студент группы ПИЖ-б-о-21-1

Гребенкин Е. А. « » 2022г.

Подпись студента\_\_\_\_\_\_\_\_\_\_\_

Работа защищена « »\_\_\_\_\_\_\_\_\_\_\_\_20\_\_г.

Проверил Воронкин Р.А. \_\_\_\_\_\_\_\_\_\_\_\_\_ (подпись)

**Цель работы:** приобретение построения приложений с интерфейсом командной строки с помощью языка программирования Python версии 3.x.

**Выполнение работы:**

1. Изучить теоретический материал работы.

2. Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT и язык программирования Python.

3. Выполните клонирование созданного репозитория.

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

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

6. Создайте проект PyCharm в папке репозитория.

7. Проработайте примеры лабораторной работы. Создайте для них отдельные модули языка Python. Зафиксируйте изменения в репозитории.

8. Приведите в отчете скриншоты результатов выполнения примера при различных исходных данных вводимых с клавиатуры.

9. Зафиксируйте сделанные изменения в репозитории.

10. Приведите в отчете скриншоты работы программ решения индивидуальных заданий.

11. Зафиксируйте сделанные изменения в репозитории.

12. Добавьте отчет по лабораторной работе в формате PDF в папку doc репозитория. Зафиксируйте изменения.

13. Выполните слияние ветки для разработки с веткой master/main.

14. Отправьте сделанные изменения на сервер GitHub.

15. Отправьте адрес репозитория GitHub на электронный адрес преподавателя.

**Проработка примера:**

#!/usr/bin/env python

# -\*- coding: utf-8 -\*-

import argparse

import json

import os.path

from datetime import date

def add\_worker(staff, name, post, year):

    """

    Добавить данные о работнике.

    """

    staff.append(

        {

            "name": name,

            "post": post,

            "year": year

        }

    )

    return staff

def display\_workers(staff):

    """

    Отобразить список работников.

    """

    # Проверить, что список работников не пуст.

    if staff:

        # Заголовок таблицы.

        line = '+-{}-+-{}-+-{}-+-{}-+'.format(

            '-' \* 4,

            '-' \* 30,

            '-' \* 20,

            '-' \* 8

        )

        print(line)

        print(

            '| {:^4} | {:^30} | {:^20} | {:^8} |'.format(

                "No",

                "Ф.И.О.",

                "Должность",

                "Год"

            )

        )

        print(line)

        # Вывести данные о всех сотрудниках.

        for idx, worker in enumerate(staff, 1):

            print(

                '| {:>4} | {:<30} | {:<20} | {:>8} |'.format(

                    idx,

                    worker.get('name', ''),

                    worker.get('post', ''),

                    worker.get('year', 0)

                )

            )

            print(line)

    else:

        print("Список работников пуст.")

def select\_workers(staff, period):

    """

    Выбрать работников с заданным стажем.

    """

    # Получить текущую дату.

    today = date.today()

    # Сформировать список работников.

    result = []

    for employee in staff:

        if today.year - employee.get('year', today.year) >= period:

            result.append(employee)

    # Возвратить список выбранных работников.

    return result

def save\_workers(file\_name, staff):

    """

    Сохранить всех работников в файл JSON.

    """

    # Открыть файл с заданным именем для записи.

    with open(file\_name, "w", encoding="utf-8") as fout:

        # Выполнить сериализацию данных в формат JSON.

        # Для поддержки кирилицы установим ensure\_ascii=False

        json.dump(staff, fout, ensure\_ascii=False, indent=4)

def load\_workers(file\_name):

    """

    Загрузить всех работников из файла JSON.

    """

    # Открыть файл с заданным именем для чтения.

    with open(file\_name, "r", encoding="utf-8") as fin:

        return json.load(fin)

def main(command\_line=None):

    # Создать родительский парсер для определения имени файла.

    file\_parser = argparse.ArgumentParser(add\_help=False)

    file\_parser.add\_argument(

        "filename",

        action="store",

        help="The data file name"

    )

    # Создать основной парсер командной строки.

    parser = argparse.ArgumentParser("workers")

    parser.add\_argument(

        "--version",

        action="version",

        version="%(prog)s 0.1.0"

    )

    subparsers = parser.add\_subparsers(dest="command")

    # Создать субпарсер для добавления работника.

    add = subparsers.add\_parser(

        "add",

        parents=[file\_parser],

        help="Add a new worker"

    )

    add.add\_argument(

        "-n",

        "--name",

        action="store",

        required=True,

        help="The worker's name"

    )

    add.add\_argument(

        "-p",

        "--post",

        action="store",

        help="The worker's post"

    )

    add.add\_argument(

        "-y",

        "--year",

        action="store",

        type=int,

        required=True,

        help="The year of hiring"

    )

    # Создать субпарсер для отображения всех работников.

    \_ = subparsers.add\_parser(

        "display",

        parents=[file\_parser],

        help="Display all workers"

    )

    # Создать субпарсер для выбора работников.

    select = subparsers.add\_parser(

        "select",

        parents=[file\_parser],

        help="Select the workers"

    )

    select.add\_argument(

        "-P",

        "--period",

        action="store",

        type=int,

        required=True,

        help="The required period"

    )

    # Выполнить разбор аргументов командной строки.

    args = parser.parse\_args(command\_line)

    # Загрузить всех работников из файла, если файл существует.

    is\_dirty = False

    if os.path.exists(args.filename):

        workers = load\_workers(args.filename)

    else:

        workers = []

        # Добавить работника.

    if args.command == "add":

        workers = add\_worker(

            workers,

            args.name,

            args.post,

            args.year

        )

        is\_dirty = True

        # Отобразить всех работников.

    elif args.command == "display":

        display\_workers(workers)

        # Выбрать требуемых рааботников.

    elif args.command == "select":

        selected = select\_workers(workers, args.period)

        display\_workers(selected)

        # Сохранить данные в файл, если список работников был изменен.

    if is\_dirty:

        save\_workers(args.filename, workers)

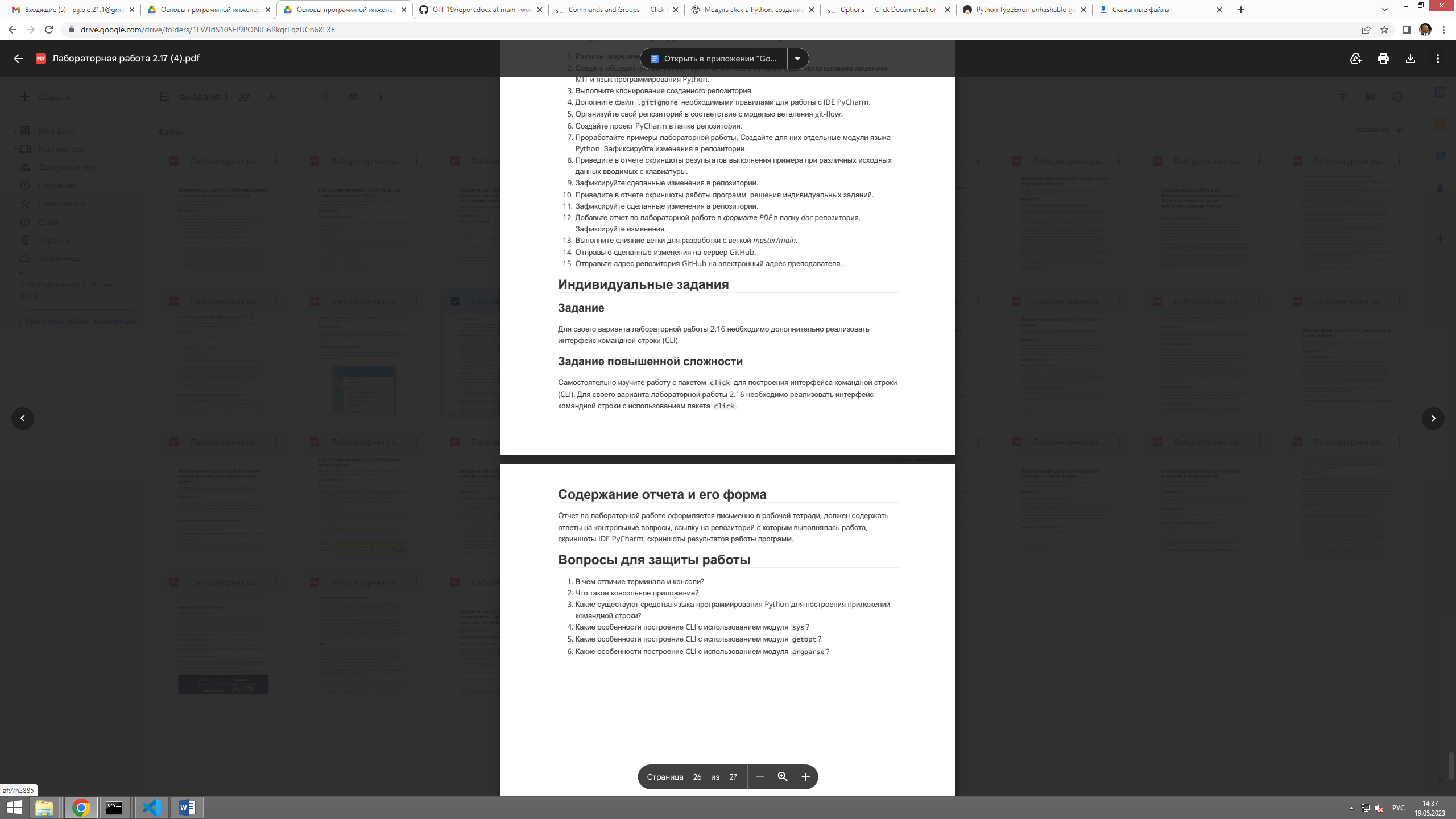
if \_\_name\_\_ == "\_\_main\_\_":

    main()

**Листинг 1** – Код для примера № 1 лабораторной работы

**Самостоятельные задания**

**Задание 1.**



#!/usr/bin/env python

# -\*- coding: utf-8 -\*-

import json

import argparse

import os.path

import sys

LETTERS = 'eyuioa'

def load\_words(file\_name: str) -> dict:

    with open(file\_name, 'r', encoding='utf-8') as f:

        return json.load(f)

def add\_words(file\_name: str, word: dict) -> None:

    with open(file\_name, 'r', encoding='utf-8') as f:

        content\_json = json.load(f)

    content\_json.append(word)

    with open(file\_name, 'w', encoding='utf-8') as f:

        json.dump(content\_json, f)

def update\_json(file\_name: str, words: list) -> list:

    for item in words:

        for letter in item["word"]:

            if letter in LETTERS:

                item["result"] = True

            else:

                item["result"] = False

    with open(file\_name, 'w', encoding='utf-8') as f:

        json.dump(words, f)

    return words

def display(words):

    if words:

        line = '+-{}-+-{}-+-{}-+'.format(

            '-' \* 4,

            '-' \* 30,

            '-' \* 20

        )

        print(line)

        print(

            "| {:^4} | {:^30} | {:^20} |".format(

                "№",

                "Слово",

                "Результат"

            )

        )

        print(line)

        for idx, word in enumerate(words, 1):

            print(

                "| {:^4} | {:^30} | {:^20} |".format(

                    idx,

                    word.get('word', ''),

                    word.get('result', ''),

                )

            )

        print(line)

    else:

        print("Список пуст.")

def main(command\_line=None):

    file\_parser = argparse.ArgumentParser(add\_help=False)

    file\_parser.add\_argument(

        "filename",

        action="store",

        help="The data file name"

    )

    parser = argparse.ArgumentParser("words")

    parser.add\_argument(

        "--version",

        action="version",

        version="%(prog)s 0.1.0"

    )

    subparsers = parser.add\_subparsers(dest="command")

    add = subparsers.add\_parser(

        "add",

        parents=[file\_parser],

        help="Add a new word"

    )

    add.add\_argument(

        "-w",

        "--word",

        action="store",

        required=True,

        help="The word instance"

    )

    add.add\_argument(

        "-r",

        "--result",

        action="store",

        required=True,

        help="Handler result"

    )

    \_ = subparsers.add\_parser(

        "display",

        parents=[file\_parser],

        help="Display all words"

    )

    \_ = subparsers.add\_parser(

        "update",

        parents=[file\_parser],

        help="Update values"

    )

    args = parser.parse\_args(command\_line)

    if os.path.exists(args.filename):

        words = load\_words(args.filename)

    else:

        with open(args.filename, "w", encoding="utf-8") as f:

            json.dump(list(), f)

            words = list()

    if args.command == "add":

        word = {

            "word": args.word,

            "result": args.result

        }

        add\_words(

            args.filename,

            word

        )

    elif args.command == "display":

        display(words)

    elif args.command == "update":

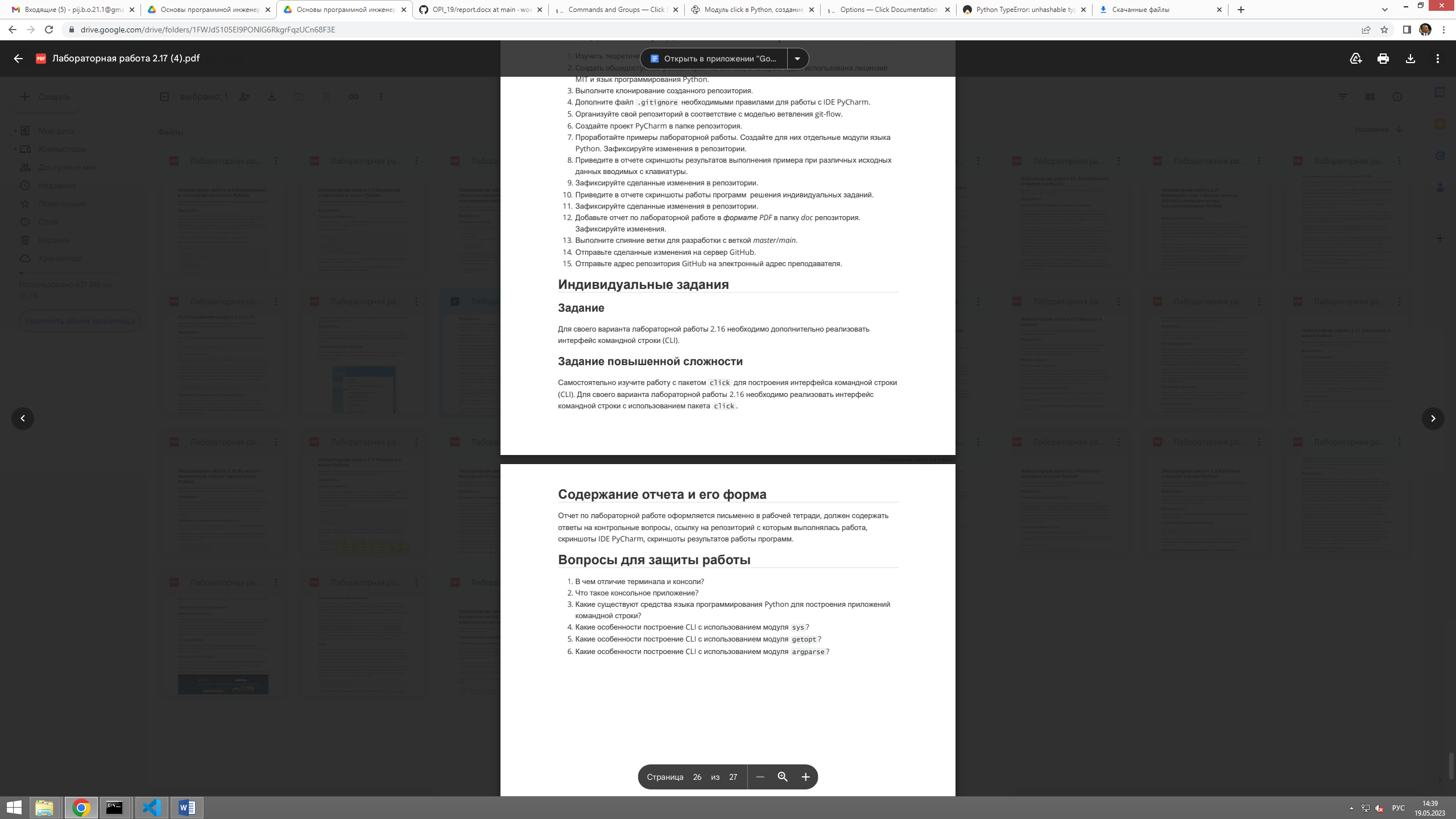
        update\_json(args.filename, words)

if \_\_name\_\_ == '\_\_main\_\_':

    main()

**Листинг 2** – Код задания для ЛР №2.17

**Задание 2.**



#!/usr/bin/env python

# -\*- coding: utf-8 -\*-

import json

import click

import os.path

import sys

LETTERS = 'eyuioa'

def load\_words(file\_name: str) -> dict:

    with open(file\_name, 'r', encoding='utf-8') as f:

        return json.load(f)

def add\_words(file\_name: str, word: dict) -> None:

    with open(file\_name, 'r', encoding='utf-8') as f:

        content\_json = json.load(f)

    content\_json.append(word)

    with open(file\_name, 'w', encoding='utf-8') as f:

        json.dump(content\_json, f)

def update\_json(file\_name: str) -> list:

    words = load\_words(file\_name)

    for item in words:

        for letter in item["word"]:

            if letter in LETTERS:

                item["result"] = True

            else:

                item["result"] = False

    with open(file\_name, 'w', encoding='utf-8') as f:

        json.dump(words, f)

    return words

def display\_json(words):

    if words:

        line = '+-{}-+-{}-+-{}-+'.format(

            '-' \* 4,

            '-' \* 30,

            '-' \* 20

        )

        print(line)

        print(

            "| {:^4} | {:^30} | {:^20} |".format(

                "№",

                "Слово",

                "Результат"

            )

        )

        print(line)

        for idx, word in enumerate(words, 1):

            print(

                "| {:^4} | {:^30} | {:^20} |".format(

                    idx,

                    word.get('word', ''),

                    word.get('result', ''),

                )

            )

        print(line)

    else:

        print("Список пуст.")

@click.command()

@click.argument("filename")

@click.option("-w", "--word", required=True, type=str)

@click.option("-r", "--result", required=True, type=int)

def add(filename, word, result):

    word = {

            "word": word,

            "result": result

        }

    add\_words(

        filename,

        word

    )

@click.command()

@click.argument("filename")

def display(filename):

    if os.path.exists(filename):

        words = load\_words(filename)

    else:

        with open(filename, "w", encoding="utf-8") as f:

            json.dump(list(), f)

            words = list()

    display\_json(words)

@click.command()

@click.argument("filename")

def update(filename):

    update\_json(filename)

@click.group()

def main():

    pass

main.add\_command(add)

main.add\_command(update)

main.add\_command(display)

if \_\_name\_\_ == '\_\_main\_\_':

    main()

**Листинг 3** – Задание повышенной сложности для ЛР 2.17

**Вопросы для защиты работы**

1. В чём отличие терминала от консоли?

*Терминал (от лат. terminus — граница) — устройство или ПО, выступающее посредником между*

*человеком и вычислительной системой. Обычно данный термин используется, когда точка*

*доступа к системе вынесена в отдельное физическое устройство и предоставляет свой*

*пользовательский интерфейс на основе внутреннего интерфейса (например, сетевых*

*протоколов).*

*Консоль console — исторически реализация терминала с клавиатурой и текстовым дисплеем. В*

*настоящее время это слово часто используется как синоним сеанса работы или окна оболочки*

*командной строки. В том же смысле иногда применяется и слово “терминал”.*

1. Что такое консольное приложение?

*Консольное приложение console application — вид ПО, разработанный с расчётом на работу*

*внутри оболочки командной строки, т.е. опирающийся на текстовый ввод-вывод.*

1. Какие существуют средства языка программирования Python для построения приложений командной строки?

*Модули: sys, getopt, argparse, click*

1. Какие особенности построение CLI с использованием модуля sys?

*Это базовый модуль, который с самого начала поставлялся с Python. Он использует подход,*

*очень похожий на библиотеку C, с использованием argc и argv для доступа к аргументам.*

*Модуль sys реализует аргументы командной строки в простой структуре списка с именем*

*sys.argv.*

*Каждый элемент списка представляет собой единственный аргумент. Первый элемент в списке*

*sys.argv [0] – это имя скрипта Python. Остальные элементы списка, от sys.argv [1] до*

*sys.argv [n] , являются аргументами командной строки с 2 по n. В качестве разделителя между*

*аргументами используется пробел. Значения аргументов, содержащие пробел, должны быть*

*заключены в кавычки, чтобы их правильно проанализировал sys .*

*Эквивалент argc – это просто количество элементов в списке. Чтобы получить это значение,*

*используйте оператор len() . Позже мы покажем это на примере кода.*

1. Какие особенности построение CLI с использованием модуля getopt?

*Как вы могли заметить ранее, модуль sys разбивает строку командной строки только на*

*отдельные фасеты. Модуль getopt в Python идет немного дальше и расширяет разделение*

*входной строки проверкой параметров. Основанный на функции C getopt , он позволяет*

*использовать как короткие, так и длинные варианты, включая присвоение значений.*

*На практике для правильной обработки входных данных требуется модуль sys . Для этого*

*необходимо заранее загрузить как модуль sys , так и модуль getopt . Затем из списка входных*

*параметров мы удаляем первый элемент списка (см. код ниже) и сохраняем оставшийся список*

*аргументов командной строки в переменной с именем arguments\_list .*

1. Какие особенности построение CLI с использованием модуля argparse?

*Начиная с версий Python 2.7 и Python 3.2, в набор стандартных библиотек была включена*

*библиотека argparse для обработки аргументов (параметров, ключей) командной строки.*

*Одним из аргументов противников включения argparse в Python был довод о том, что в*

*стандартных модулях и без этого содержится две библиотеки для семантической обработки*

*(парсинга) параметров командной строки. Однако, как заявляют разработчики argparse ,*

*библиотеки getopt и optparse уступают argparse по нескольким причинам:*

*обладая всей полнотой действий с обычными параметрами командной строки, они не умеют*

*обрабатывать позиционные аргументы (positional arguments). Позиционные аргументы —*

*это аргументы, влияющие на работу программы, в зависимости от порядка, в котором они в*

*эту программу передаются. Простейший пример — программа cp, имеющая минимум 2*

*таких аргумента («cp source destination»).*

*argparse дает на выходе более качественные сообщения о подсказке при минимуме*

*затрат (в этом плане при работе с optparse часто можно наблюдать некоторую*

*избыточность кода);*

*argparse дает возможность программисту устанавливать для себя, какие символы*

*являются параметрами, а какие нет. В отличие от него, optparse считает опции с*

*синтаксисом наподобие "-pf, -file, +rgb, /f и т.п. «внутренне противоречивыми» и «не*

*поддерживается optpars 'ом и никогда не будет»;*

*argparse даст Вам возможность использовать несколько значений переменных у одного*

*аргумента командной строки (nargs);*

*argparse поддерживает субкоманды (subcommands). Это когда основной парсер отсылает*

*к другому (субпарсеру), в зависимости от аргументов на входе.*