

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 №5
дисциплины «Основы программной инженерии»

Выполнил:
Мелтонян Одиссей
2 курс, группа ПИЖ-б-о-22-1,
09.03.04 «Программная инженерия»,
направленность (профиль) «Разработка
и сопровождение программного
обеспечения», очная форма обучения

(подпись)

Проверил:
Кандидат технических наук, доцент
кафедры инфокоммуникаций
Воронкин Р. А

(подпись)

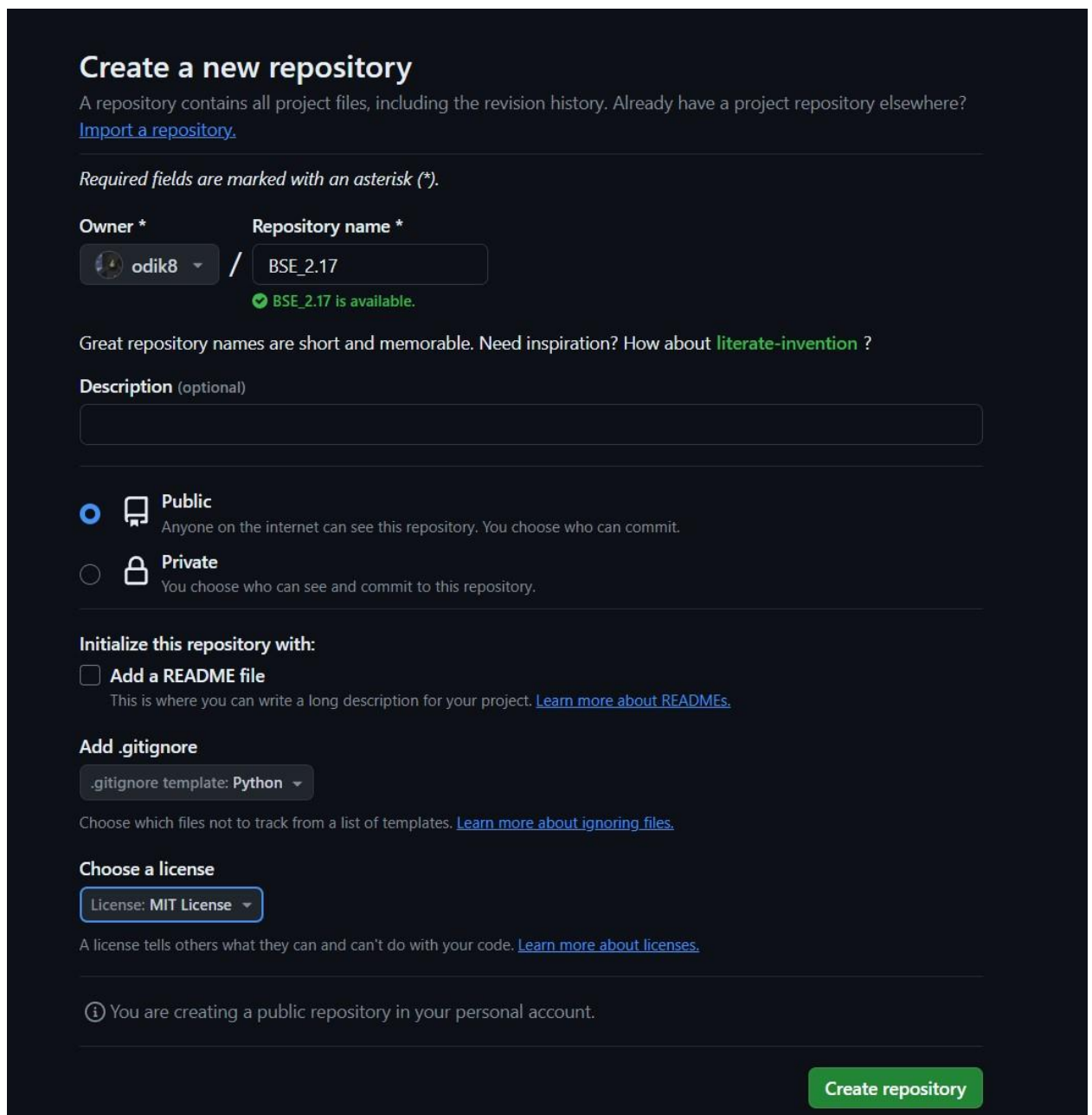
Отчет защищен с оценкой _____ Дата защиты _____

Тема: Разработка приложений с интерфейсом командной строки (CLI) в Python3

Цель работы: приобретение построения приложений с интерфейсом командной строки.

Ход работы:

1. Изучен теоретический материал работы.
2. Создан общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT 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.](#)

Required fields are marked with an asterisk ().*


Owner * **Repository name ***


 odik8 / BSE_2.17

✔ BSE_2.17 is available.

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

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: **Python**

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

Choose a license

License: **MIT License**

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

 You are creating a public repository in your personal account.

[Create repository](#)

Рисунок 1 – Создание репозитория

3. Выполнено клонирование созданного репозитория.

```
varfe@DESKTOP-E108KEN MINGW64 ~/Рабочий стол/4 семестр/ОПИ/ЛР5
$ git clone https://github.com/odik8/BSE_2.17.git
```

Рисунок 2 – Клонирование репозитория

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

5. Организован репозиторий в соответствие с моделью ветвления git-flow.

```
PS C:\Users\varfe\Рабочий стол\4 семестр\ОПИ\ЛР5\BSE_2.17> git flow init
warning: ignoring broken ref refs/heads/desktop.ini

Which branch should be used for bringing forth production releases?
warning: ignoring broken ref refs/heads/desktop.ini
- main
Branch name for production releases: [main]
warning: ignoring broken ref refs/heads/desktop.ini
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/varfe/Рабочий стол/4 семестр/ОПИ/ЛР5/BSE_2.17/.git/hooks]
```

Рисунок 3 – Инициализация git-flow

6. Выполнено индивидуальное задание. Для своего варианта лабораторной работы 2.16 необходимо дополнительно реализовать интерфейс командной строки (CLI).

Код:

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

import argparse
import json
import os.path

from jsonschema import ValidationError, validate

def add_person(
    contact_list: list,
```

```

name: str,
lastname: str,
phone: str,
birthdate: str
):
    """
    Add a new person
    """

    contact_list.append(
        {
            "name": name,
            "lastname": lastname,
            "phone": phone,
            "birthdate": birthdate,
        }
    )

def display_contact_list(contact_list: list):
    """
    Displays contact list
    """

    if contact_list:
        line = "+-{}-+-{}-+-{}-+-{}-+-{}-+ ".format(
            "-" * 4,
            "-" * 20,
            "-" * 20,
            "-" * 20,
            "-" * 20,
            "-" * 10,
        )
        print(line)
        print(
            "| {:^4} | {:^20} | {:^20} | {:^20} | {:^10} | ".format(
                "№",
                "Name",
                "Lastname",
                "Phone numbers",
                "Birth date",
            )
        )
        print(line)

        for idx, person in enumerate(contact_list, 1):
            print(
                "| {:^4} | {:>20} | {:>20} | {:>20} | {:>10} | ".format(
                    idx,
                    person.get("name", ""),
                    person.get("lastname", ""),
                    person.get("phone", ""),

```

```

        person.get("birthdate", ""),
    )
    )
    print(line)
else:
    print("Contact list is empty.")

def select_person(contact_list: list, phone: str):
    """
    Displays person by phone numbers
    """

    selected_person = ""
    for person in contact_list:
        if person.get("phone") == phone:
            selected_person = person
            print(selected_person)

def save_contact_list(file_name, contact_list):
    """
    Save contact list into JSON file.
    """
    with open(file_name, "w", encoding="utf-8") as fout:
        json.dump(contact_list, fout, ensure_ascii=False, indent=4)
    print("Data successfully saved to file", file_name)

def load_contact_list_json(file_name):
    """
    Load contact list from json
    """
    with open(file_name, "r", encoding="utf-8") as f:
        document = json.load(f)

    if all(list(map(lambda x: check_validation_json(x), document))):
        return document
    else:
        None

def check_validation_json(file_name):
    with open("schema.json") as fs:
        schema = json.load(fs)

    try:
        validate(instance=file_name, schema=schema)
        return True
    except ValidationError:
        return False

```

```

def main():
    file_parser = argparse.ArgumentParser(add_help=False)
    file_parser.add_argument("filename", help="The data file name")

    parser = argparse.ArgumentParser("Contact list")
    parser.add_argument(
        "--version",
        action="version",
        version="% (prog)s 0.1.0",
    )

    subparsers = parser.add_subparsers(dest="command")

    # Subparser for adding a new person
    add = subparsers.add_parser(
        "add",
        parents=[file_parser],
        help="Add a new person",
    )

    add.add_argument("-n", "--name", required=True, help="The person's name")

    add.add_argument(
        "-ln", "--lastname", required=True, help="The person's lastname"
    )

    add.add_argument(
        "-ph", "--phone", required=True, help="The person's phone numbers"
    )

    add.add_argument(
        "-bd", "--birthdate", required=True, help="The person's birth date"
    )

    # Subparser for displaying contact list
    _ = subparsers.add_parser(
        "display", parents=[file_parser], help="Display contact list"
    )

    # Subparser for selecting a person
    select = subparsers.add_parser(
        "select", parents=[file_parser], help="Selet a person by phone numbers"
    )

    select.add_argument(
        "-p", "--phone", required=True, help="The required phone numbers"
    )

    # Parse command line arguments

```

```

args = parser.parse_args()

# Load contact list from a file if the file exists.
is_dirty = False

if os.path.exists(args.filename):
    contact_list = load_contact_list_json(args.filename)
else:
    contact_list = []

match args.command:
    case "add":
        add_person(
            contact_list,
            args.name,
            args.lastname,
            args.phone,
            args.birthdate,
        )
        is_dirty = True

    case "display":
        display_contact_list(contact_list)

    case "select":
        select_person(contact_list, args.phone)

if is_dirty:
    save_contact_list(args.filename, contact_list)

if __name__ == "__main__":
    main()

```

Работа программы:

```

PS C:\Users\varfe\Рабочий стол\4 семестр\ОПИ\ЛР5\BSE_2.17\code> python .\task_1.py -h
usage: Contact list [-h] [--version] {add,display,select} ...

positional arguments:
  {add,display,select}
    add                Add a new person
    display            Display contact list
    select             Select a person by phone numbers

options:
  -h, --help          show this help message and exit
  --version            show program's version number and exit

```

Рисунок 4 – Помощь

```
PS C:\Users\varfe\Рабочий стол\4 семестр\ОПИ\ЛР5\BSE_2.17\code> python .\task_1.py add data.json -n Ivan -ln Ivanov -ph 8999999999 -bd 01.01.2000
Data successfully saved to file data.json
PS C:\Users\varfe\Рабочий стол\4 семестр\ОПИ\ЛР5\BSE_2.17\code>
```

Рисунок 5 – Добавление нового человека

```
(bse_2.17) C:\Users\varfe\Рабочий стол\4 семестр\ОПИ\ЛР5\BSE_2.17\code>python task_1.py display data.json
```

%	Name	Lastname	Phone numbers	Birth date
1	odissey	meltonyan	123123	01.01.1999
2	igor	vikhorkov	777777	07.07.1977
3	Yaroslav	Mindal	099099	12.12.2012
4	Ivan	Ivanov	8999999999	12.01.2003

Рисунок 6 – Отображение списка контактов

(Данные у последнего были изменены)

```
(bse_2.17) C:\Users\varfe\Рабочий стол\4 семестр\ОПИ\ЛР5\BSE_2.17\code>python task_1.py select data.json -p 123123
{'name': 'odissey', 'lastname': 'meltonyan', 'phone': '123123', 'birthdate': '01.01.1999'}
(bse_2.17) C:\Users\varfe\Рабочий стол\4 семестр\ОПИ\ЛР5\BSE_2.17\code>
```

Рисунок 7 – Выбор по номеру телефона

7. Выполнено задание повышенной сложности. Самостоятельно изучите работу с пакетом click для построения интерфейса командной строки (CLI). Для своего варианта лабораторной работы 2.16 необходимо реализовать интерфейс командной строки с использованием пакета click .

Код:

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

import json
import os.path

import click
from jsonschema import ValidationError, validate

def add_person(
    contact_list,
    name: str,
    lastname: str,
    phone: str,
    birthdate: str
):
    """
    Add a new person
    """
```



```

contact_list.append(
    {
        "name": name,
        "lastname": lastname,
        "phone": phone,
        "birthdate": birthdate
    }
)

def display_contact_list(contact_list):
    """
    Displays contact list
    """

    if contact_list:
        line = "+-{}-+-{}-+-{}-+-{}-+-{}-+".format(
            "-" * 4,
            "-" * 20,
            "-" * 20,
            "-" * 20,
            "-" * 20,
            "-" * 10,
        )
        print(line)
        print(
            "| {:^4} | {:^20} | {:^20} | {:^20} | {:^10} |".format(
                "№",
                "Name",
                "Lastname",
                "Phone numbers",
                "Birth date",
            )
        )
        print(line)

        for idx, person in enumerate(contact_list, 1):
            print(
                "| {:^4} | {:>20} | {:>20} | {:>20} | {:>10} |".format(
                    idx,
                    person.get("name", ""),
                    person.get("lastname", ""),
                    person.get("phone", ""),
                    person.get("birthdate", ""),
                )
            )
            print(line)
        else:
            print("Contact list is empty.")

def select_person(contact_list, phone: str):

```

```

"""
Displays person by phone numbers
"""

selected_person = ""
for person in contact_list:
    if person.get("phone") == phone:
        selected_person = person

print(selected_person)

def save_contact_list(file_name, contact_list):
    """
    Save contact list into JSON file.
    """
    with open(file_name, "w", encoding="utf-8") as fout:
        json.dump(contact_list, fout, ensure_ascii=False, indent=4)
    print("Data successfully saved to file", file_name)

def load_contact_list_json(file_name):
    """
    Load contact list from json
    """
    with open(file_name, "r", encoding="utf-8") as f:
        document = json.load(f)

    if all(list(map(lambda x: check_validation_json(x), document))):
        return document
    else:
        print("Invalid data in the JSON file.")
        print(document) # Добавим эту строку для отладочной информации
        return None

def check_validation_json(file_name):
    with open("schema.json") as fs:
        schema = json.load(fs)

    try:
        validate(instance=file_name, schema=schema)
        return True
    except ValidationError:
        return False

@click.group()
def cli():
    pass

```

```

@cli.command()
@click.argument("filename")
@click.option("-n", "--name", required=True, help="The person's name")
@click.option("-ln", "--lastname", required=True, help="The person's lastname")
@click.option(
    "-ph", "--phone", required=True, help="The person's phone numbers"
)
@click.option(
    "-bd", "--birthdate", required=True, help="The person's birth date"
)
def add(filename, name, lastname, phone, birthdate):
    """
    Add a new person
    """
    if os.path.exists(filename):
        contact_list = load_contact_list_json(filename)
        contact_list.append(
            add_person(contact_list, name, lastname, phone, birthdate)
        )
        save_contact_list(filename, contact_list)
        click.echo(f"{name} {lastname} added")
    else:
        click.echo("Invalid path entered.")

@cli.command()
@click.argument("filename")
def display(filename):
    """
    Displays contact list
    """
    if os.path.exists(filename):
        contact_list = load_contact_list_json(filename)
        display_contact_list(contact_list)
    else:
        click.echo("Invalid path entered.")

@cli.command()
@click.argument("filename")
@click.option("-p", "--phone", required=True)
def select(filename, phone):
    """
    Selet a person by phone numbers
    """
    if os.path.exists(filename):
        contact_list = load_contact_list_json(filename)
        select_person(contact_list, phone)
    else:
        click.echo("PathError: Invalid path entered.")

```

```
if __name__ == "__main__":
    cli()
```

Работа программы:

```
(bse_2.17) C:\Users\varfe\Рабочий стол\4 семестр\ОПИ\ЛР5\BSE_2.17\code>python hardtask.py
Usage: hardtask.py [OPTIONS] COMMAND [ARGS]...

Options:
  --help  Show this message and exit.

Commands:
  add       Add a new person
  display   Displays contact list
  select    Selet a person by phone numbers

(bse_2.17) C:\Users\varfe\Рабочий стол\4 семестр\ОПИ\ЛР5\BSE_2.17\code>|
```

Рисунок 8 – Помощь

```
(bse_2.17) C:\Users\varfe\Рабочий стол\4 семестр\ОПИ\ЛР5\BSE_2.17\code>python hardtask.py add data.json -n Petr -ln Petr
ov -ph 897777777777 -bd 19.08.2005
Data successfully saved to file data.json
Petr Petrov added

(bse_2.17) C:\Users\varfe\Рабочий стол\4 семестр\ОПИ\ЛР5\BSE_2.17\code>|
```

Рисунок 9 – Добавление контакта

```
(bse_2.17) C:\Users\varfe\Рабочий стол\4 семестр\ОПИ\ЛР5\BSE_2.17\code>python hardtask.py display data.json
```

#	Name	Lastname	Phone numbers	Birth date
1	odissey	meltonyan	123123	01.01.1999
2	igor	vikhorkov	777777	07.07.1977
3	Yaroslav	Minda1	099099	12.12.2012
4	Ivan	Ivanov	89999999999	12.01.2003
5	Petr	Petrov	89777777777	19.08.2005

Рисунок 10 – Отображение списка

```
(bse_2.17) C:\Users\varfe\Рабочий стол\4 семестр\ОПИ\ЛР5\BSE_2.17\code>python hardtask.py select data.json -p 777777
{'name': 'igor', 'lastname': 'vikhorkov', 'phone': '777777', 'birthdate': '07.07.1977'}
```

Рисунок 11 – Выбор по номеру телефона

8. Зафиксированы сделанные изменения в репозитории.
9. Добавлен отчет по лабораторной работе в формате PDF в папку doc репозитория.

Вопросы для защиты работы:

1. В чем отличие терминала и консоли? – Терминал и консоль оба представляют интерфейс командной строки для взаимодействия с операционной системой. Однако, есть различия:

- Терминал обычно является программой, которая предоставляет доступ к командной строке. Он может быть графическим или текстовым.

- Консоль обычно относится к текстовому интерфейсу, предоставляемому операционной системой для ввода команд напрямую. Например, в Windows это может быть командная строка CMD, в Linux/Unix - терминал в режиме текстовой консоли.

2. Что такое консольное приложение? – Консольное приложение — это программа, которая запускается и выполняется в командной строке (консоли). Она взаимодействует с пользователем через текстовый интерфейс, часто используя ввод с клавиатуры и вывод на экран.

3. Какие существуют средства языка программирования Python для построения приложений командной строки? – В Python существует несколько средств для построения приложений командной строки:

- Модуль ``sys`` предоставляет доступ к некоторым системным переменным и функциям, что может быть полезно для обработки аргументов командной строки.

- Модуль ``getopt`` используется для парсинга аргументов командной строки в стиле POSIX (Portable Operating System Interface), то есть в стиле, совместимом с большинством UNIX-подобных операционных систем.

- Модуль ``argparse`` предоставляет более мощные и удобные средства для обработки аргументов командной строки, позволяя определять ожидаемые аргументы, их типы, поддерживает генерацию справки и другие функции.

4. Особенности построения CLI с использованием модуля ``sys``:

- Модуль ``sys.argv`` позволяет получить список аргументов командной строки, переданных при запуске скрипта.

- Недостатком является то, что обработка аргументов может быть не очень удобной, особенно при наличии сложных аргументов с ключами.

5. Особенности построения CLI с использованием модуля ``getopt``:

- ``getopt`` позволяет определять короткие и длинные опции с соответствующими значениями.
- Он позволяет более гибко обрабатывать аргументы, чем ``sys.argv``, но все же требует ручной обработки.

6. Особенности построения CLI с использованием модуля ``argparse``:

- ``argparse`` позволяет определять ожидаемые аргументы, их типы, ограничения и даже генерировать справку для пользователя.
- Он автоматически обрабатывает типы аргументов и проверяет их на корректность, что делает его более удобным и безопасным в использовании.