

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 №3
дисциплины «Анализ данных»
Вариант 29

Выполнил:
Саенко Андрей Максимович
2 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и
вычислительная техника»,
направленность (профиль)
«Программное обеспечение средств
вычислительной техники и
автоматизированных систем», очная
форма обучения

(подпись)

Руководитель практики:
Воронкин Р.А., канд. технических
наук, доцент кафедры
инфокоммуникаций

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____
Ставрополь, 2024 г.

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

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

Порядок выполнения работы:

Проработаны примеры из лабораторной работы:

Пример 1. Для примера 1 лабораторной работы 2.16 разработайте интерфейс командной строки.

Код программы:

```
#!/usr/bin/env python3
# -*- 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(
            "№",
            "Ф.И.О.",
            "Должность",
            "Год"
        ))
```

```

        "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 кириллицы 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(

```

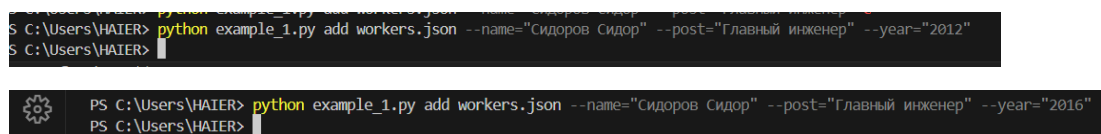
```

        "-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 работников.
    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()

```

Результат работы программы:



The screenshot shows a Windows command prompt window with the following commands and output:

```

PS C:\Users\HAIER> python example_1.py add workers.json --name="Сидоров Сидор" --post="Главный инженер" --year="2012"
PS C:\Users\HAIER>
PS C:\Users\HAIER> python example_1.py add workers.json --name="Сидоров Сидор" --post="Главный инженер" --year="2016"
PS C:\Users\HAIER>

```

Рисунок 1 – Добавление работников

```
example_1.py  {} workers.json X
Users > HAIER > {} workers.json > ...
[
  {
    "name": "Сидоров Сидор",
    "post": "Главный инженер",
    "year": 2012
  }
]
```

Рисунок 2 – Содержимое файла workers.json

№	Ф.И.О.	Должность	Год
1	Сидоров Сидор	Главный инженер	2012
2	Сидоров Сидор	Главный инженер	2016

PS C:\Users\HAIER>

Рисунок 3 – Вывод данных из файла workers.json

```
PS C:\Users\HAIER> python example_1.py select workers.json --period=12
+-----+-----+-----+-----+
| № | Ф.И.О. | Должность | Год |
+-----+-----+-----+-----+
| 1 | Сидоров Сидор | Главный инженер | 2012 |
+-----+-----+-----+-----+
PS C:\Users\HAIER>
```

Рисунок 4 – Вывод некоторых данных из файла workers.json

Выполнены задания:

Вариант 29

Задание 1. Для своего варианта лабораторной работы 2.16 необходимо дополнительно реализовать интерфейс командной строки (CLI).

Код программы:

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

import argparse
import json
import os.path
```

```

from datetime import date

def add_route(staff, start, end, number):
    """
    Добавить маршрут
    """
    staff.append({
        'name_start': start,
        'name_end': end,
        'number': number
    })
    return staff

def list(routes):
    """
    Вывести список маршрутов
    """
    if routes:
        line = '+-{}-+-{}-+-{}-+-{}-+'.format(
            '-' * 4,
            '-' * 30,
            '-' * 30,
            '-' * 8
        )
        print(line)

        print('| {:^4} | {:^30} | {:^30} | {:^8} |'.format(
            "№",
            "Начальный пункт",
            "Конечный пункт",
            "Номер"
        ))
        print(line)

        for idx, route in enumerate(routes, 1):
            print('| {:>4} | {:<30} | {:<30} | {:>8} |'.format(
                idx,
                route.get('name_start', ""),
                route.get('name_end', ""),
                route.get('number', 0)
            ))
            print(line)
    else:
        print("Список маршрутов пуст.")

def save_routes(file_name, staff):
    """
    Сохранить все маршруты в файл JSON.
    """
    with open(file_name, "w", encoding="utf-8") as fout:

```

```

# Выполнить сериализацию данных в формат JSON.
# Для поддержки кириллицы установим ensure_ascii=False
json.dump(staff, fout, ensure_ascii=False, indent=4)

def load_routes(file_name):
    """
    Загрузить все маршруты из файла JSON.
    """
    with open(file_name, "r", encoding="utf-8") as fin:
        return json.load(fin)

def select_routes(routes, command):
    """
    Вывести выбранные маршруты
    """
    station = command
    count = 0

    for route in routes:
        if (station.lower() == route["name_start"].lower() or
            station.lower() == route["name_end"].lower()):

            count += 1
            print('{:>4}: {}-{}, номер маршрута: {}'.format(count,
                route["name_start"], route["name_end"], route["number"]))

    if count == 0:
        print("Маршрут не найден.")

def main(command_line=None):
    """
    Основная функция
    """
    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("routes")
    parser.add_argument(
        "--version",
        action="version",
        version="% (prog)s 0.1.0"
    )
    subparsers = parser.add_subparsers(dest="command")
    # Создать субпарсер для добавления маршрута.
    add = subparsers.add_parser(
        "add",
        parents=[file_parser],
        help="Add a new route"
    )

```



```

)
add.add_argument(
    "-s",
    "--start",
    action="store",
    required=True,
    help="Start position on route"
)
add.add_argument(
    "-e",
    "--end",
    action="store",
    help="End position on route"
)
add.add_argument(
    "-n",
    "--number",
    action="store",
    type=int,
    required=True,
    help="Number of route"
)
# Создать субпарсер для отображения всех маршрутов.
_ = subparsers.add_parser(
    "display",
    parents=[file_parser],
    help="Display all routes"
)
# Создать субпарсер для выбора маршрутов.
select = subparsers.add_parser(
    "select",
    parents=[file_parser],
    help="Select the routes"
)
select.add_argument(
    "-t",
    "--station",
    action="store",
    type=str,
    required=True,
    help="Routes with this station"
)
# Выполнить разбор аргументов командной строки.
args = parser.parse_args(command_line)
# Загрузить все маршруты из файла, если файл существует.
is_dirty = False
if os.path.exists(args.filename):
    routes = load_routes(args.filename)
else:
    routes = []
# Добавить маршрут.
if args.command == "add":

```

```

if(routes==None):
    routes = []
routes = add_route(
    routes,
    args.start,
    args.end,
    args.number
)
is_dirty = True
# Отобразить все маршруты.
elif args.command == "display":
    list(routes)
# Выбрать требуемые маршруты.
elif args.command == "select":
    select_routes(routes, args.station)
# Сохранить данные в файл, если список маршрутов был изменен.
if is_dirty:
    save_routes(args.filename, routes)

if __name__ == '__main__':
    main()

```

Результат работы программы:

```

PS C:\Users\HAIER> python ind_1.py add routes.json --start="Москва" --end="Воронеж" --number=1
PS C:\Users\HAIER> python ind_1.py add routes.json --start="Норильск" --end="Воронеж" --number=2
PS C:\Users\HAIER> python ind_1.py add routes.json --start="Москва" --end="Ярославль" --number=3
PS C:\Users\HAIER> python ind_1.py display routes.json

```

№	Начальный пункт	Конечный пункт	Номер
1	Москва	Воронеж	1
2	Норильск	Воронеж	2
3	Москва	Ярославль	3

Рисунок 5 – Добавление маршрутов в файл routes.json и вывод его содержимого

```

PS C:\Users\HAIER> python ind_1.py select routes.json --station="Москва"
1: Москва-Воронеж, номер маршрута: 1
2: Москва-Ярославль, номер маршрута: 3

```

Рисунок 6 – Вывод маршрутов, у которых начальной или конечной точкой служит Москва

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 программы:

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

# 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

import click
import json
import os.path

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

@commands.command("add")
@click.argument("filename")
@click.option("--start", help="Start station")
@click.option("--end", help="End station")
@click.option("--number", type=int, help="Number of route")
def add_route(filename, start, end, number):
    """
    Добавить маршрут
    """

    routes = load_routes(filename)
    route = {
        'name_start': start,
        'name_end': end,
        'number': number
    }
    routes.append(route)
    save_routes(filename, routes)

def list(routes):
    """
    Вывести список маршрутов
    """
    if routes:
        line = '+-{}-+-{}-+-{}-+-{}-+'.format(
            '-' * 4,
            '-' * 30,
            '-' * 30,
            '-' * 8
        )
        print(line)
```

```

print('| {:.^4} | {:.^30} | {:.^30} | {:.^8} |'.format(
    "№",
    "Начальный пункт",
    "Конечный пункт",
    "Номер"
)
)
print(line)

for idx, route in enumerate(routes, 1):
    print('| {:.>4} | {:.<30} | {:.<30} | {:.>8} |'.format(
        idx,
        route.get('name_start', ''),
        route.get('name_end', ''),
        route.get('number', 0)
    )
)
print(line)
else:
    print("Список маршрутов пуст.")

def save_routes(file_name, staff):
    """
    Сохранить все маршруты в файл JSON.
    """
    with open(file_name, "w", encoding="utf-8") as fout:
        # Выполнить сериализацию данных в формат JSON.
        # Для поддержки кириллицы установим ensure_ascii=False
        json.dump(staff, fout, ensure_ascii=False, indent=4)

def load_routes(file_name):
    """
    Загрузка маршрутов из файла JSON
    """
    if os.path.isfile(file_name):
        with open(file_name, "r", encoding="utf-8") as fin:
            return json.load(fin)
    return []

@commands.command("display")
@click.argument("filename")
def display_routes(filename):
    """
    Отобразить список маршрутов.
    """
    routes = load_routes(filename)
    list(routes)

@commands.command("select")
@click.argument("filename")

```

```

@click.option("--station", help="Start or end station")
def select_routes(filename, station):
    """
    Вывести выбранные маршруты
    """
    st = station
    routes = load_routes(filename)
    count = 0
    result = []
    for route in routes:
        if (st.lower() == route["name_start"].lower() or
            st == route["name_end"].lower()):
            result.append(route)
            count += 1

    if count == 0:
        print("Маршрут не найден.")
    else:
        list(result)

def main():
    """
    Основная функция
    """
    commands()

if __name__ == '__main__':
    main()

```

Результат работы программы:

```

PS C:\Users\HAIER> python hard_1.py add routes.json --start="Нью-Йорк" --end="Омск" --number=5
PS C:\Users\HAIER> python hard_1.py select routes.json --station="Нью-Йорк"
+-----+-----+-----+
| № | Начальный пункт | Конечный пункт | Номер |
+-----+-----+-----+
| 1 | Нью-Йорк | Омск | 5 |
+-----+-----+-----+
PS C:\Users\HAIER> python hard_1.py display routes.json
+-----+-----+-----+
| № | Начальный пункт | Конечный пункт | Номер |
+-----+-----+-----+
| 1 | Москва | Воронеж | 1 |
+-----+-----+-----+
| 2 | Норильск | Воронеж | 2 |
+-----+-----+-----+
| 3 | Москва | Ярославль | 3 |
+-----+-----+-----+
| 4 | Нью-Йорк | Омск | 5 |
+-----+-----+-----+
PS C:\Users\HAIER> 

```

Рисунок 6 – Результат работы программы

```
[
  {
    "name_start": "Москва",
    "name_end": "Воронеж",
    "number": 1
  },
  {
    "name_start": "Норильск",
    "name_end": "Воронеж",
    "number": 2
  },
  {
    "name_start": "Москва",
    "name_end": "Ярославль",
    "number": 3
  },
  {
    "name_start": "Нью-Йорк",
    "name_end": "Омск",
    "number": 5
  }
]
```

Рисунок 7 – Содержимое файла routes.json

Ответы на контрольные вопросы:

1. В чем отличие терминала и консоли?

Консоль - исторически реализация терминала с клавиатурой и текстовым дисплеем.

Терминал - устройство или ПО, выступающее посредником между человеком и вычислительной системой.

2. Что такое консольное приложение?

Консольное приложение - вид ПО, разработанный с расчётом на работу внутри оболочки командной строки, т.е. опирающийся на текстовый ввод-вывод.

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

Существуют следующие средства: sys, getopt, argparse, click.

4. Какие особенности построение CLI с использованием модуля sys ?

Модуль sys с точки зрения имен и использования имеет прямое отношение к библиотеке C (libc). Модуль sys реализует аргументы командной строки в простой структуре списка с именем sys.argv .

5. Какие особенности построение CLI с использованием модуля getopt ?

Основанный на функции C getopt , он позволяет использовать как короткие, так и длинные варианты, включая присвоение значений.

6. Какие особенности построение CLI с использованием модуля argparse ?

Модуль argparse позволяет проводить анализ аргументов sys.argv, конвертировать строковых аргументов в объекты программы и работать с ними; форматировать и выводить информативные подсказки.

Выводы: в ходе выполнения работы получены навыки написания программ с интерфейсом командной строки.