

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 №2.17
дисциплины «Анализ данных»
Вариант 13

Выполнил:
Иващенко Олег Андреевич
2 курс, группа ИВТ-б-о-22-1,
09.03.02 «Информационные и
вычислительные машины»,
направленность (профиль)
«Программное обеспечение
средств вычислительной техники
и автоматизированных систем»

(подпись)

Руководитель практики:
Воронкин Роман Александрович,
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 навыков построения приложений с интерфейсом командной строки с помощью языка программирования Python версии 3.x.

Порядок выполнения работы

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

Таблица 1 – Код программы example.py:

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

import argparse
import json
import os.path
import sys
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"
        )
    )
    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()

```

```

PS C:\Users\UnnamedUser\OneDrive\Документы\СФУ\Python\Analysis_2.17\exec> python example.py add data.json --name="Сидор
ов Сидор" --post="Главный инженер" --year=2012

```

Рисунок 1.1 – Добавление новой записи в файл data.json

```

[
  {
    "name": "Иванов Иван",
    "post": "Директор",
    "year": 2007
  },
  {
    "name": "Петров Петр",
    "post": "Бухгалтер",
    "year": 2010
  },
  {
    "name": "Сидоров Сидор",
    "post": "Главный инженер",
    "year": 2012
  }
]

```

Рисунок 1.2 – Содержимое файла data.json

```

PS C:\Users\UnnamedUser\OneDrive\Документы\СФУ\Python\Analysis_2.17\exec> python example.py display data.json

```

%	Ф.И.О.	Должность	Год
1	Иванов Иван	Директор	2007
2	Петров Петр	Бухгалтер	2010
3	Сидоров Сидор	Главный инженер	2012

Рисунок 1.3 – Вывод содержимого файла data.json

```
PS C:\Users\UnnamedUser\OneDrive\Документы\СКОУ\Python\Analysis_2.17\exec> python example.py select data.json --period=12
```

%	Ф.И.О.	Должность	Год
1	Иванов Иван	Директор	2007

Рисунок 1.4 – Выборка из файла

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

Таблица 2 – Код программы

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

import json
import sys
from datetime import datetime
import argparse
import os.path

def print_help():
    """
    Функция вывода доступных пользователю команд
    """

    print("list - вывод всех добавленных записей")
    print("add - добавление новых записей")
    print("find - найти запись по фамилии")
    print("exit - завершение работы программы")

def add_worker(workers, surname, name, phone, date):
    """
    Функция добавления новой записи, возвращает запись
    """

    workers.append(
        {
            "surname": surname,
            'name': name,
            'phone': phone,
            'date': date
        }
    )

    return workers

def print_list(list):
    """
```

```
Функция выводит на экран список всех существующих записей
"""
```

```
for member in list:
    print(f"{member['surname']} {member['name']} | "
          f"{member['phone']} | {member['date']}")
```

```
def find_member(workers, period):
    """
```

```
Функция для вывода на экран всех записей, чьи фамилии совпадают
с введённой (не возвращает никаких значений)
"""
```

```
count = 0
members = []
```

```
for member in workers:
    year = datetime.strptime(member['date'], "%d.%m.%Y").year
    if datetime.now().year - period >= year:
        members.append(member)
        count += 1
```

```
if count == 0:
    print("Записи не найдены")
else:
    return members
```

```
def save_file(filename, data):
    """
```

```
Сохранение списка сотрудников в файл формата JSON
"""
```

```
with open(filename, "w", encoding="utf-8") as file:
    json.dump(data, file, ensure_ascii=False, indent=4)
```

```
def load_file(filename):
    """
```

```
Загрузка данных о сотрудниках из указанного JSON-файла
"""
```

```
with open(filename, "r", encoding="utf-8") as file:
    return json.load(file)
```

```
def parse_datetime(value):
```

```
try:
    return datetime.strptime(value, "%d.%m.%Y")
except ValueError:
```



```

        print("Error")

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("workers")
    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 worker"
    )
    add.add_argument(
        "-s",
        "--surname",
        action="store",
        required=True,
        help="The worker's surname"
    )
    add.add_argument(
        "-n",
        "--name",
        action="store",
        required=True,
        help="The worker's name"
    )
    add.add_argument(
        "-p",
        "--phone",
        action="store",
        help="The worker's phone"
    )
    add.add_argument(
        "-d",
        "--date",
        action="store",
        required=True,
        help="The date of hiring"
    )

```

```

_ = subparsers.add_parser(
    "display",
    parents=[file_parser],
    help="Display all workers"
)

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"
)

args = parser.parse_args(command_line)

is_dirty = False
if os.path.exists(args.filename):
    workers = load_file(args.filename)
else:
    workers = []

if args.command == "add":
    workers = add_worker(
        workers,
        args.surname,
        args.name,
        args.phone,
        args.date
    )
    is_dirty = True

elif args.command == "display":
    print_list(workers)

elif args.command == "select":
    selected = find_member(workers, args.period)
    print_list(selected)

if is_dirty:
    save_file(args.filename, workers)

if __name__ == "__main__":
    """

```

Основная программа

```
""  
main()
```

```
PS C:\Users\UnnamedUser\OneDrive\Документы\СКФУ\Python\Analysis_2.17\exec> python individual.py display ind.json  
Петров Петр | 78889994545 | 01.01.1995  
Иванов Иван | 79998881212 | 10.10.1998
```

Рисунок 2.1 – Вывод изначального списка

```
PS C:\Users\UnnamedUser\OneDrive\Документы\СКФУ\Python\Analysis_2.17\exec> python individual.py add -s="Сидоров" -n="Сидор" -p="78889996351" -d="10.10.2020" ind.json  
PS C:\Users\UnnamedUser\OneDrive\Документы\СКФУ\Python\Analysis_2.17\exec> python individual.py display ind.json  
Петров Петр | 78889994545 | 01.01.1995  
Иванов Иван | 79998881212 | 10.10.1998  
Сидоров Сидор | 78889996351 | 10.10.2020
```

Рисунок 2.2 – Добавление новой записи и вывод списка

```
PS C:\Users\UnnamedUser\OneDrive\Документы\СКФУ\Python\Analysis_2.17\exec> python individual.py display ind.json  
Петров Петр | 78889994545 | 01.01.1995  
Иванов Иван | 79998881212 | 10.10.1998  
Сидоров Сидор | 78889996351 | 10.10.2020  
PS C:\Users\UnnamedUser\OneDrive\Документы\СКФУ\Python\Analysis_2.17\exec> python individual.py select -p="20" ind.json  
Петров Петр | 78889994545 | 01.01.1995  
Иванов Иван | 79998881212 | 10.10.1998  
PS C:\Users\UnnamedUser\OneDrive\Документы\СКФУ\Python\Analysis_2.17\exec>
```

Рисунок 2.3 – Выборка из списка

Задание повышенной сложности. Самостоятельно изучить 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`.

Таблица 3 – Код программы

```
#!/usr/bin/env python3  
# -*- coding: utf-8 -*-  
  
import json  
import sys  
from datetime import datetime  
import argparse  
import os.path  
import click  
  
@click.group()  
def cli():  
    pass  
  
@cli.command()  
@click.option('--filename', '-f', help='Имя файла')  
def display(filename):  
    """  
    Функция (click) вывода списка существующих записей из указанного файла  
    """  
  
    try:
```

```

        workers = load_file(filename)
        print_list(workers)
        print()
    except:
        print("Ошибка в имени файла\n")

@cli.command()
@click.option('--filename', '-f', help='Имя файла')
@click.option("--surname", "-s", help="Фамилия работника")
@click.option("--name", "-n", help="Имя работника")
@click.option("--phone", "-p", help="Номер телефона работника")
@click.option("--date", "-d", help="Дата приёма работника")
def add(filename, surname, name, phone, date):
    """
    Функция (click) добавления в указанный файл новой записи
    """

    workers = load_file(filename)
    workers = add_worker(workers, surname, name, phone, date)
    save_file(filename, workers)

@cli.command()
@click.option('--filename', '-f', help='Имя файла-источника')
@click.option('--period', '-p', type=int, help='Искомый период (лет)')
def select(filename, period):
    """
    Функция (click) выбора записей из указанного файла по периоду
    """

    try:
        workers = find_member(load_file(filename), period)
        print_list(workers)
        print()
    except TypeError:
        print("Записи не найдены\n")

def add_worker(workers, surname, name, phone, date):
    """
    Функция добавления новой записи, возвращает запись
    """

    workers.append(
        {
            "surname": surname,
            'name': name,
            'phone': phone,
            'date': date
        }
    )

    return workers

```

```

def print_list(list):
    """
    Функция выводит на экран список всех существующих записей
    """

    for member in list:
        print(f"{member['surname']} {member['name']} | "
              f"{member['phone']} | {member['date']}")

def find_member(workers, period):
    """
    Функция для вывода на экран всех записей, чьи фамилии совпадают
    с введённой (не возвращает никаких значений)
    """

    count = 0
    members = []

    for member in workers:
        year = datetime.strptime(member['date'], "%d.%m.%Y").year
        if datetime.now().year - period >= year:
            members.append(member)
            count += 1

    if count != 0:
        return members

def save_file(filename, data):
    """
    Сохранение списка сотрудников в файл формата JSON
    """

    with open(filename, "w", encoding="utf-8") as file:
        json.dump(data, file, ensure_ascii=False, indent=4)

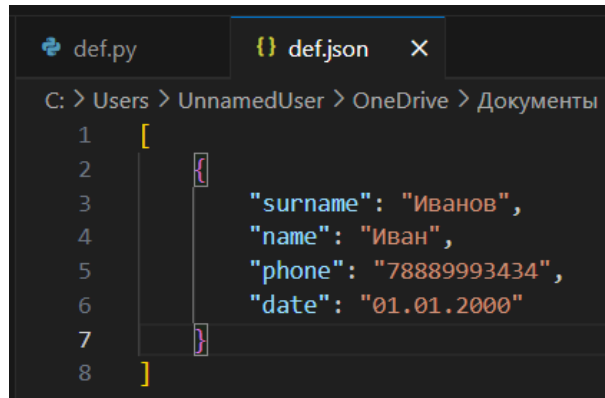
def load_file(filename):
    """
    Загрузка данных о сотрудниках из указанного JSON-файла
    """

    with open(filename, "r", encoding="utf-8") as file:
        return json.load(file)

def parse_datetime(value):
    try:
        return datetime.strptime(value, "%d.%m.%Y")
    except ValueError:
        print("Error")

```

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



```
[
  {
    "surname": "Иванов",
    "name": "Иван",
    "phone": "78889993434",
    "date": "01.01.2000"
  }
]
```

Рисунок 3.1 – Изначальное содержимое файла def.json

```
PS C:\Users\UnnamedUser\OneDrive\Документы\СКФУ\Python\Analysis_2.17\exec> python def.py display -f def.json
Иванов Иван | 78889993434 | 01.01.2000
```

Рисунок 3.2 – Вывод содержимого файла в консоль

```
PS C:\Users\UnnamedUser\OneDrive\Документы\СКФУ\Python\Analysis_2.17\exec> python def.py add --help
Usage: def.py add [OPTIONS]

Функция (click) добавления в указанный файл новой записи

Options:
  -f, --filename TEXT  Имя файла
  -s, --surname TEXT   Фамилия работника
  -n, --name TEXT      Имя работника
  -p, --phone TEXT     Номер телефона работника
  -d, --date TEXT      Дата приёма работника
  --help              Show this message and exit.
```

Рисунок 3.3 – Вывод подсказки команды add

```
PS C:\Users\UnnamedUser\OneDrive\Документы\СКФУ\Python\Analysis_2.17\exec> python def.py add -f def.json -s Петров -n
Петр -p 78886665373 -d 20.10.2005
PS C:\Users\UnnamedUser\OneDrive\Документы\СКФУ\Python\Analysis_2.17\exec> python def.py display -f def.json
Иванов Иван | 78889993434 | 01.01.2000
Петров Петр | 78886665373 | 20.10.2005
```

Рисунок 3.4 – Добавление новой записи

```
PS C:\Users\UnnamedUser\OneDrive\Документы\СКФУ\Python\Analysis_2.17\exec> python def.py display -f def.json
Иванов Иван | 78889993434 | 01.01.2000
Петров Петр | 78886665373 | 20.10.2005

PS C:\Users\UnnamedUser\OneDrive\Документы\СКФУ\Python\Analysis_2.17\exec> python def.py select -f def.json -p 20
Иванов Иван | 78889993434 | 01.01.2000
```

Рисунок 3.5 – Выборка записей по периоду

Контрольные вопросы

1. В чём отличие терминала и консоли?

Терминал – 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 – 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».

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?

`sys` – модуль, предоставляющий доступ к некоторым переменным и функциям, специфичными для интерпретатора Python. Хотя `sys` в первую очередь используется для управления интерпретатором и его окружением, его так же можно использовать для обработки аргументов командной строки, переданных при запуске скрипта, с помощью парса `sys.argv`.

`getopt` – модуль, предоставляющий функции для разработки аргументов командной строки в стиле Unix, когда аргументы могут быть короткими (однобуквенными) или длинными (полнословными). Этот модуль предоставляет функцию `getopt.getopt`, которая позволяет определить список опций и их значения, переданных скрипту через аргументы командной строки.

`argparse` – стандартная библиотека Python, предназначенная для разбора аргументов командной строки и создания интерфейса командной строки. Она позволяет легко определять аргументы, поддерживает различные типы аргументов и автоматически генерирует справку для пользователя.

4. Какие особенности построения CLI с использованием модуля `sys`?

Использование `sys.argv` предоставляет простой способ доступа к аргументам командной строки без необходимости импортировать дополнительные библиотеки. Однако нужно работать напрямую с аргументами командной строки, поэтому требуется вручную проверять и обрабатывать аргументы. Это может привести к большому объёму кода и более сложному управлению ошибками.

5. Какие особенности построения CLI с использованием модуля `getopt`?

Модуль `getopt` позволяет определять и использовать как короткие (однобуквенные), так и длинные (полные слова) опции командной строки. Но подобно `sys.argv`, `getopt` требует ручного разбора аргументов. Это может быть утомительным и привести к большому количеству кода, особенно при обработке различных опций и аргументов.

6. Какие особенности построения CLI с использованием модуля `argparse`?

Модуль `argparse` предоставляет высокоуровневый API для определения и обработки аргументов командной строки. Можно определять ожидаемые аргументы и опции с помощью декларативного синтаксиса, что делает код более читаемым и поддерживаемым. `argparse` автоматически генерирует справку на основе определённых аргументов и опций. Это упрощает документирование программы и делает её более доступной для пользователей. Также `argparse` позволяет определять различные типы аргументов, такие как целые числа, строки и другие. Она также обеспечивает проверку корректности введённых значений, что может сократить количество ошибок и упростить отладку.

Выводы: В процессе выполнения лабораторной работы были приобретены навыки построения приложений с интерфейсом командной строки с помощью языка программирования Python, был изучен пакет `click`, позволяющий другими методами создать интерфейс CLI. Был проработан пример, выполнено индивидуальное задание и задание повышенной сложности.