

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 9**

Выполнил:  
Дудкин Константин Александрович  
2 курс, группа ИВТ-б-о-22-1,  
09.03.01 «Информатика и  
вычислительная техника»  
направление «Программное  
обеспечение средств вычислительной  
техники и автоматизированных  
систем»,  
очная форма обучения

---

(подпись)

Руководитель практики:  
Кандидат технических наук, доцент  
кафедры инфокоммуникаций, доцент  
Воронкин Роман Александрович

---

(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

Проработал пример:

```

def main(command_line=None):
    # Создать родительский парсер для определения имени файла.
    file_parser = argparse.ArgumentParser(add_help=False)
    file_parser.add_argument(
        *name_or_flags: "filename",
        action="store",
        help="The data file name"
    )

    # Создать основной парсер командной строки.
    parser = argparse.ArgumentParser("workers")
    parser.add_argument(
        *name_or_flags: "--version",
        action="version",
        version="%(prog)s 0.1.0"
    )

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

    # Создать субпарсер для добавления работника.
    add = subparsers.add_parser(
        name: "add",
        parents=[file_parser],
        help="Add a new worker"
    )

    add.add_argument(
        *name_or_flags: "-n",
        "--name",
        action="store",
        required=True,
        help="The worker's name"
    )

    add.add_argument(
        *name_or_flags: "-p",
        "--post",
        action="store",
        help="The worker's post"
    )

```

Рисунок 1. Измененный код примера (часть 1)

```

131     )
132     add.add_argument(
133         *name_or_flags: "-y",
134         "--year",
135         action="store",
136         type=int,
137         required=True,
138         help="The year of hiring"
139     )
140     # Создать субпарсер для отображения всех работников.
141     _ = subparsers.add_parser(
142         name: "display",
143         parents=[file_parser],
144         help="Display all workers"
145     )
146     # Создать субпарсер для выбора работников.
147     select = subparsers.add_parser(
148         name: "select",
149         parents=[file_parser],
150         help="Select the workers"
151     )
152     select.add_argument(
153         *name_or_flags: "-P",
154         "--period",
155         action="store",
156         type=int,
157         required=True,
158         help="The required period"
159     )
160     # Выполнить разбор аргументов командной строки.
161     args = parser.parse_args(command_line)
162     # Загрузить всех работников из файла, если файл существует.

```

Рисунок 2. Измененный код примера (часть 2)

```

163     is_dirty = False
164     if os.path.exists(args.filename):
165         workers = load_workers(args.filename)
166     else:
167         workers = []
168     # Добавить работника.
169     if args.command == "add":
170         workers = add_worker(
171             workers,
172             args.name,
173             args.post,
174             args.year
175         )
176         is_dirty = True
177     # Отобразить всех работников.
178     elif args.command == "display":
179         display_workers(workers)
180     # Выбрать требуемых работников.
181     elif args.command == "select":
182         selected = select_workers(workers, args.period)
183         display_workers(selected)
184     # Сохранить данные в файл, если список работников был изменен.
185     if is_dirty:
186         save_workers(args.filename, workers)
187

```

Рисунок 3. Измененный код примера (часть 3)

Выполнил индивидуальное задание: Нужно в программе из прошлой лабораторной работы реализовать интерфейс командной строки:

```

def main(command_line=None):
    # Родительский парсер для определения имени файла.
    file_parser = argparse.ArgumentParser(add_help=False)
    file_parser.add_argument(
        *name_or_flags: "filename",
        action="store",
        help="Имя файла с данными"
    )

    # Основной парсер командной строки.
    parser = argparse.ArgumentParser("routes")
    parser.add_argument(
        *name_or_flags: "--version",
        action="version",
        version="%(prog)s 0.1.0"
    )
    subparsers = parser.add_subparsers(dest="command")

    # Субпарсер для добавления маршрута.
    add_parser = subparsers.add_parser(
        name: "add",
        parents=[file_parser],
        help="Добавить новый маршрут"
    )
    add_parser.add_argument(
        *name_or_flags: "--first",
        action="store",
        required=True,
        help="Место отправки"
    )
    add_parser.add_argument(
        *name_or_flags: "--second",
        action="store",
        required=True,
        help="Место прибытия"
    )

    # Субпарсер для отображения всех маршрутов.
    display_parser = subparsers.add_parser(
        name: "list",
        parents=[file_parser],
        help="Показать список маршрутов"
    )

```

Рисунок 4. Измененный код индивидуального задания (часть 1)

```

# Субпарсер для экспорта маршрутов.
export_parser = subparsers.add_parser(
    name="export",
    parents=[file_parser],
    help="Экспортировать маршруты в JSON-файл"
)
export_parser.add_argument(
    *name_or_flags: "export_file",
    action="store",
    help="Имя файла для экспорта"
)

# Субпарсер для импорта маршрутов.
import_parser = subparsers.add_parser(
    name="import",
    parents=[file_parser],
    help="Импортировать маршруты из JSON-файла"
)
import_parser.add_argument(
    *name_or_flags: "import_file",
    action="store",
    help="Имя файла для импорта"
)

# Разбор аргументов командной строки.
args = parser.parse_args(command_line)

# Список маршрутов
routes = []

# Загрузить маршруты из файла, если файл существует.
if os.path.exists(args.filename):
    routes = import_json(args.filename)

elif args.command != "import":
    # Создать пустой файл, если он не существует при любой ком
    with open(args.filename, 'w', encoding='utf-8') as file:
        json.dump(obj: [], file)

```

Рисунок 5. Измененный код индивидуального задания (часть 2)

```

# Добавить маршрут.
if args.command == "add":
    add_route(routes, args.first, args.second)
    export_to_json(args.filename, routes)

# Показать список маршрутов.
elif args.command == "list":
    list_of_routes(routes)

# Экспортировать маршруты в файл.
elif args.command == "export":
    export_to_json(args.export_file, routes)

# Импортировать маршруты из файла.
elif args.command == "import":
    routes = import_json(args.import_file)
    export_to_json(args.filename, routes)

```

Рисунок 6. Измененный код индивидуального задания (часть 3)

```

(base) [code_raider@archlinux Progs]$ python individual.py add individual.json --first "Ставрополь" --second "Ессентуки"
(base) [code_raider@archlinux Progs]$ python individual.py add individual.json --first "Пятигорск" --second "Москва"
(base) [code_raider@archlinux Progs]$ 

```



```

[
  {
    "first": "Ставрополь",
    "second": "Ессентуки"
  },
  {
    "first": "Пятигорск",
    "second": "Москва"
  }
]

```

Рисунок 7. Проверка работоспособности программы

Ответы на вопросы:

1. Терминал — программа, которая обеспечивает взаимодействие пользователя с командной строкой. Он может быть как графическим, так и текстовым, и имеет внедренное распознавание команд от обычного текста

Консоль — окно, в котором отображается командная строка ОС. Является частью терминала, ведь именно благодаря консоли и обеспечивается взаимодействие пользователя и компьютером



2. Консольное приложение — программа, запускаемая и выполняемая в командной строке. Они обычно не имеют графического интерфейса, а значит выполняются обычно в текстовом режиме

3. Средства Python для построения приложений командной строки:

- `argparse` — модуль, предоставляющий простые инструменты для создания интерфейса командной строки

- `getopt` — низкоуровневый вариант `argparse`, обеспечивающий парсинг опций командной строки в стиле Unix

- `sys.argv` — стандартный список аргументов командной строки в Python

4. `sys.argv` включает в себя аргументы командной строки, которые запускаются вместе со скриптом. Парсинг аргументов в нем должен быть реализован вручную с использованием методов строк и списков

5. `getopt` поддерживает короткие и длинные опции с корректным разбором аргументов, однако работа с ним значительно сложнее, чем с `argparse`

6. `argparse` включает в себя расширенные и удобные инструменты для создания интерфейса командной строки. Он автоматически генерирует справку на основе определенных аргументов, поддерживает многие типы аргументов и их проверку и легко интегрируется с кодом программы в ясный синтаксис для определения аргументов

Вывод: В ходе данной лабораторной работы были приобретены навыки создания интерфейса командной строки с помощью языка программирования Python версии 3.x.