МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
ИНСТИТУТ ЦИФРОВОГО РАЗВИТИЯ

Отчет о лабораторной работе №5 по дисциплине основы программной инженерии

Выполнил:
Выходцев Егор Дмитриевич,
2 курс, группа ПИЖ-б-о-20-1,

Проверил:
Доцент кафедры инфокоммуникаций,
Воронкин Р.А.

Ставрополь, 2022 г

1. Примеры из методических указаний

ex1.py

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

# Include standard modules
import getopt, sys


if __name__ == '__main__':
    full_cmd_arguments = sys.argv
    # Keep all but the first
    argument_list = full_cmd_arguments[1:]
    print(argument_list)
    short_options = "ho:v"
    long_options = ["help", "output=", "verbose"]
    try:
        arguments, values = getopt.getopt(argument_list, short_options,
                                          long_options)
    except getopt.error as err:
        # Output error, and return with an error code
        print(str(err))
        sys.exit(2)
    # Evaluate given options
    for current_argument, current_value in arguments:
        if current_argument in ("-v", "--verbose"):
            print("Enabling verbose mode")
        elif current_argument in ("-h", "--help"):
            print("Displaying help")
        elif current_argument in ("-o", "--output"):
            print(f"Enabling special output mode ({current_value})")

```

```
Windows PowerShell
(C) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

PS C:\Users\Evil\PycharmProjects\LR#5> python ex1.py -h
['-h']
Displaying help
PS C:\Users\Evil\PycharmProjects\LR#5> python ex.py --output=green --help -v
E:\Python\python.exe: can't open file 'C:\Users\Evil\PycharmProjects\LR#5\ex.py': [Errno 2] No such file or directory
PS C:\Users\Evil\PycharmProjects\LR#5> python ex1.py --output=green --help -v
['--output=green', '--help', '-v']
Enabling special output mode (green)
Displaying help
Enabling verbose mode
PS C:\Users\Evil\PycharmProjects\LR#5> python ex1.py -verbose
['-verbose']
option -e not recognized
PS C:\Users\Evil\PycharmProjects\LR#5>
```

ex1.py ex2.py

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

import argparse


if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument(
        "square", type=int, help="display a square of a given number"
    )
    parser.add_argument(
        "-v", "--verbose", action="store_true",
        help="increase output verbosity"
    )
    args = parser.parse_args()
    answer = args.square ** 2
    if args.verbose:
        print("the square of {} equals {}".format(args.square, answer))
    else:
        print(answer)

```

```
PS C:\Users\Evil\PycharmProjects\LR#5> python ex2.py 4
16
PS C:\Users\Evil\PycharmProjects\LR#5> python ex2.py 3 7 12
usage: ex2.py [-h] [-v] square
ex2.py: error: unrecognized arguments: 7 12
PS C:\Users\Evil\PycharmProjects\LR#5> python ex2.py 12
144
PS C:\Users\Evil\PycharmProjects\LR#5>
```

```
PS C:\Users\Evil\PycharmProjects\LR#5> python ex2.py 7 -v
the square of 7 equals 49
PS C:\Users\Evil\PycharmProjects\LR#5>
```

ex1.py ex2.py ex3.py

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

import argparse


if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument("echo")
    args = parser.parse_args()
    print(args.echo)

```

```
PS C:\Users\Evil\PycharmProjects\LR#5> python ex3.py hello
hello
PS C:\Users\Evil\PycharmProjects\LR#5> python ex3.py 37 56
usage: ex3.py [-h] echo
ex3.py: error: unrecognized arguments: 56
PS C:\Users\Evil\PycharmProjects\LR#5>
```

ex1.py ex2.py ex3.py ex4.py

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

import argparse


if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument(
        "square",
        type=int,
        help="display a square of a given number"
    )
    parser.add_argument(
        "-v",
        "--verbosity",
        type=int,
        help="increase output verbosity"
    )
    args = parser.parse_args()
    answer = args.square ** 2
    if args.verbosity == 2:
        print("the square of {} equals {}".format(args.square, answer))
    elif args.verbosity == 1:
        print("{}^2 == {}".format(args.square, answer))
    else:
        print(answer)

```

```
PS C:\Users\Evil\PycharmProjects\LR#5> python ex4.py 7
49
PS C:\Users\Evil\PycharmProjects\LR#5> python ex4.py 12 -v 1
12^2 == 144
PS C:\Users\Evil\PycharmProjects\LR#5> python ex4.py 13 --verbosity 2
the square of 13 equals 169
PS C:\Users\Evil\PycharmProjects\LR#5>
```

ex1.py ex2.py ex3.py ex4.py ex5.py

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

import argparse


if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument(
        "square",
        type=int,
        help="display the square of a given number"
    )
    parser.add_argument(
        "-v",
        "--verbosity",
        action="count",
        help="increase output verbosity"
    )
    args = parser.parse_args()
    answer = args.square ** 2
    if args.verbosity == 2:
        print("the square of {} equals {}".format(args.square, answer))
    elif args.verbosity == 1:
        print("{}^2 == {}".format(args.square, answer))
    else:
        print(answer)

```

```
PS C:\Users\Evil\PycharmProjects\LR#5> python ex5.py 4 -v
4^2 == 16
PS C:\Users\Evil\PycharmProjects\LR#5> python ex5.py 9 -vv
the square of 9 equals 81
PS C:\Users\Evil\PycharmProjects\LR#5>
```

ex1.py ex2.py ex3.py ex4.py ex5.py ex6.py

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

import argparse


if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument("x", type=int, help="the base")
    parser.add_argument("y", type=int, help="the exponent")
    parser.add_argument("-v", "--verbosity", action="count", default=0)
    args = parser.parse_args()
    answer = args.x ** args.y
    if args.verbosity >= 2:
        print("{} to the power {} equals {}".format(args.x, args.y, answer))
    elif args.verbosity >= 1:
        print("{}^{} == {}".format(args.x, args.y, answer))
    else:
        print(answer)

```

```
PS C:\Users\Evil\PycharmProjects\LR#5> python ex6.py 2 9
512
PS C:\Users\Evil\PycharmProjects\LR#5> python ex6.py 3 10
59049
PS C:\Users\Evil\PycharmProjects\LR#5>
```

examp1.py

```python
#!/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(
                "No",
```

examp1.py

```
                    "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):
    """
```

examp1.py

```
    """
    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"
    )
```

examp1.py

```
        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,
```

examp1.py

```
            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()

```

```
PS C:\Users\Evil\PycharmProjects\LR#5> python .\examp1.py add data.json --name="Hfhf hdjd" --post="NHdjdjd" --year=2016
PS C:\Users\Evil\PycharmProjects\LR#5> python .\examp1.py add data.json --name="Gdydusdjs" --post="Ejskam" --year=2018
PS C:\Users\Evil\PycharmProjects\LR#5> python .\examp1.py display data.json
+------+-------------------------------+----------------------+----------+
|  No  |            Ф.И.О.             |      Должность       |   Год    |
+------+-------------------------------+----------------------+----------+
|    1 | Hfhf hdjd                     | NHdjdjd              |     2016 |
|    2 | Gdydusdjs                     | Ejskam               |     2018 |
+------+-------------------------------+----------------------+----------+
```

2. Индивидуальное задание

ind1.py

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

import argparse
import json
import os.path


def get_flight(fls, dest, num, type):
    """
    Добавить данные о работнике.
    """
    fls.append(
        {
            "flight_destination": dest,
            "flight_number": num,
            "airplane_type": type
        }
    )
    return fls


def display_flights(flights):
    """
     Отобразить список рейсов
    """
    if flights:
        line = '+-{}-+-{}-+-{}-+-{}-+'.format(
            '-' * 4,
            '-' * 30,
            '-' * 20,
            '-' * 15
        )
        print(line)
        print(
            '| {:^4} | {:^30} | {:^20} | {:^15} |'.format(
                "No",
                "Пункт назначения",
                "Номер рейса",
```

display_flights() › if flights › for idx, flight in enumerate(fl...

ind1.py

```
                    "Номер рейса",
                    "Тип самолета"
                )
            )
            print(line)
            for idx, flight in enumerate(flights, 1):
                print(
                    '| {:>4} | {:<30} | {:<20} | {:<15} |'.format(
                        idx,
                        flight.get('flight_destination', ''),
                        flight.get('flight_number', ''),
                        flight.get('airplane_type', 0)
                    )
                )
            print(line)

        else:
            print("Список рейсов пуст")


def select_flights(flights, airplane_type):
    """
    Выбрать рейсы самолётов заданного типа
    """
    count = 0
    res = []
    for flight in flights:
        if flight.get('airplane_type') == airplane_type:
            count += 1
            res.append(flight)
    if count == 0:
        print("рейсы не найдены")

    return res


def save_flights(file_name, fls):
    """
    Сохранить все записи полётов в файл JSON.
```

display_flights() › if flights › for idx, flight in enumerate(fl...

ind1.py

```
    Сохранить все записи полётов 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(fls, fout, ensure_ascii=False, indent=4)


def load_flights(file_name):
    """
    Загрузить все записи полётов 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):
    """
    Главная функция программы
    """
    file_parser = argparse.ArgumentParser(add_help=False)
    file_parser.add_argument(
        "filename",
        action="store",
        help="The data file name"
    )
    parser = argparse.ArgumentParser("flights")
    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 flight"
```

ind1.py

```
        help="Add a new flight"
    )
    add.add_argument(
        "-fld",
        "--flight_dest",
        action="store",
        required=True,
        help="The flight destination"
    )
    add.add_argument(
        "-n",
        "--number",
        action="store",
        help="The flight number"
    )
    add.add_argument(
        "-t",
        "--type",
        action="store",
        required=True,
        help="The airplane type"
    )
    _ = subparsers.add_parser(
        "display",
        parents=[file_parser],
        help="Display all flights"
    )
    select = subparsers.add_parser(
        "select",
        parents=[file_parser],
        help="Select the flights"
    )
    select.add_argument(
        "-t",
        "--type",
        action="store",
        required=True,
        help="The required flight type"
    )
```

display_flights() › if flights › for idx, flight in enumerate(fl...

```python
    )
    args = parser.parse_args(command_line)
    is_dirty = False
    if os.path.exists(args.filename):
        flights = load_flights(args.filename)
    else:
        flights = []
    if args.command == "add":
        flights = get_flight(
            flights,
            args.flight_dest,
            args.number,
            args.type
        )
        is_dirty = True
    elif args.command == "display":
        display_flights(flights)
    elif args.command == "select":
        selected = select_flights(flights, args.type)
        display_flights(selected)
    if is_dirty:
        save_flights(args.filename, flights)


if __name__ == '__main__':
    main()

```

```
PS C:\Users\Evil\PycharmProjects\LR#5> python ind1.py add ind_data.json -fld="Thailand" -n="TH130" -t="Passenger"
PS C:\Users\Evil\PycharmProjects\LR#5> python ind1.py add ind_data.json -fld="Moscow" -n="MO234" -t="Military"
PS C:\Users\Evil\PycharmProjects\LR#5> python ind1.py display ind_data.json
+------+------------------------------+--------------------+----------------+
|  No  |       Пункт назначения       |     Номер рейса    |  Тип самолета  |
+------+------------------------------+--------------------+----------------+
|    1 | Thailand                     | TH130              | Passenger      |
|    2 | Moscow                       | MO234              | Military       |
+------+------------------------------+--------------------+----------------+
```

```
PS C:\Users\Evil\PycharmProjects\LR#5> python ind1.py select ind_data.json --type="Military"
+------+------------------------------+--------------------+----------------+
|  No  |       Пункт назначения       |     Номер рейса    |  Тип самолета  |
+------+------------------------------+--------------------+----------------+
|    1 | Moscow                       | MO234              | Military       |
+------+------------------------------+--------------------+----------------+
PS C:\Users\Evil\PycharmProjects\LR#5>
```

3. Задание повышенной сложности

click_ind.py

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

import click
import json
import os.path


def get_flight(fls, dest, num, type):
    """
    Добавить данные о работнике.
    """
    fls.append(
        {
            "flight_destination": dest,
            "flight_number": num,
            "airplane_type": type
        }
    )
    return fls


def display_flights(flights):
    """
     Отобразить список рейсов
    """
    if flights:
        line = '+-{}-+-{}-+-{}-+-{}-+'.format(
            '-' * 4,
            '-' * 30,
            '-' * 20,
            '-' * 15
        )
        print(line)
        print(
            '| {:^4} | {:^30} | {:^20} | {:^15} |'.format(
                "No",
                "Пункт назначения",
                "Номер рейса",
```

main()

click_ind.py

```
                    "Тип самолета"
                )
            )
            print(line)
            for idx, flight in enumerate(flights, 1):
                print(
                    '| {:>4} | {:<30} | {:<20} | {:<15} |'.format(
                        idx,
                        flight.get('flight_destination', ''),
                        flight.get('flight_number', ''),
                        flight.get('airplane_type', 0)
                    )
                )
            print(line)

        else:
            print("Список рейсов пуст")


def select_flights(flights, airplane_type):
    """
    Выбрать рейсы самолётов заданного типа
    """
    count = 0
    res = []
    for flight in flights:
        if flight.get('airplane_type') == airplane_type:
            count += 1
            res.append(flight)
    if count == 0:
        print("рейсы не найдены")

    return res


def save_flights(file_name, fls):
    """
    Сохранить все записи полётов в файл JSON.
    """
```

main()

click_ind.py

```
79          # Открыть файл с заданным именем для записи.
80          with open(file_name, "w", encoding="utf-8") as fout:
81              # Выполнить сериализацию данных в формат JSON.
82              # Для поддержки кирилицы установим ensure_ascii=False
83              json.dump(fls, fout, ensure_ascii=False, indent=4)
84
85
86      def load_flights(file_name):
87          """
88          Загрузить все записи полётов из файла JSON.
89          """
90          # Открыть файл с заданным именем для чтения.
91          with open(file_name, "r", encoding="utf-8") as fin:
92              return json.load(fin)
93
94
95      @click.command()
96      @click.argument('command')
97      @click.argument('filename')
98      @click.option('--flight_dest')
99      @click.option('--number')
100     @click.option('--type')
101     def main(command, filename, flight_dest, number, type):
102         """
103         Главная функция программы
104         """
105         is_dirty = False
106         if os.path.exists(filename):
107             flights = load_flights(filename)
108         else:
109             flights = []
110         if command == "add":
111             flights = get_flight(
112                 flights,
113                 flight_dest,
114                 number,
115                 type
116             )
117             is_dirty = True
```

main()

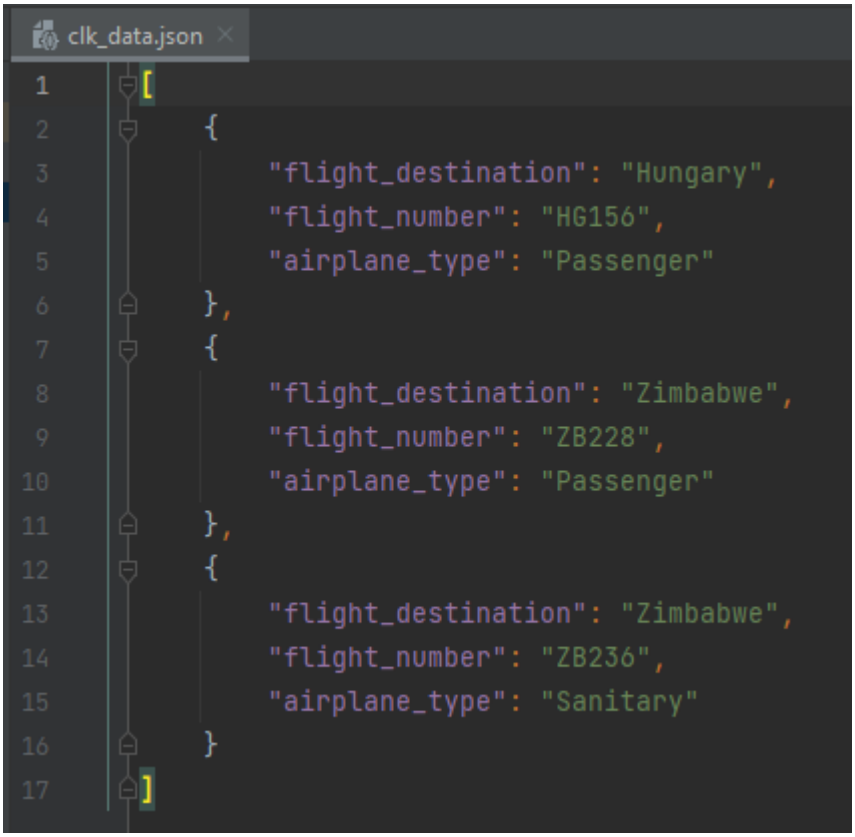
```
    elif command == "display":
        display_flights(flights)
    elif command == "select":
        selected = select_flights(flights, type)
        display_flights(selected)
    if is_dirty:
        save_flights(filename, flights)


if __name__ == '__main__':
    main()

```

```
PS C:\Users\Evil\PycharmProjects\LR#5> python .\click_ind.py add clk_data.json --flight_dest="Hungary" --number="HG156" --type="Passenger"
PS C:\Users\Evil\PycharmProjects\LR#5> python .\click_ind.py display clk_data.json
+------+-------------------------------+----------------------+-----------------+
|  No  |        Пункт назначения       |     Номер рейса      |  Тип самолета   |
+------+-------------------------------+----------------------+-----------------+
|    1 | Hungary                       | HG156                | Passenger       |
+------+-------------------------------+----------------------+-----------------+
PS C:\Users\Evil\PycharmProjects\LR#5> python .\click_ind.py add clk_data.json --flight_dest="Zimbabwe" --number="ZB228" --type="Passenger"
PS C:\Users\Evil\PycharmProjects\LR#5> python .\click_ind.py display clk_data.json
+------+-------------------------------+----------------------+-----------------+
|  No  |        Пункт назначения       |     Номер рейса      |  Тип самолета   |
+------+-------------------------------+----------------------+-----------------+
|    1 | Hungary                       | HG156                | Passenger       |
|    2 | Zimbabwe                      | ZB228                | Passenger       |
+------+-------------------------------+----------------------+-----------------+
PS C:\Users\Evil\PycharmProjects\LR#5> python .\click_ind.py add clk_data.json --flight_dest="Zimbabwe" --number="ZB236" --type="Sanitary"
PS C:\Users\Evil\PycharmProjects\LR#5> python .\click_ind.py display clk_data.json
+------+-------------------------------+----------------------+-----------------+
|  No  |        Пункт назначения       |     Номер рейса      |  Тип самолета   |
+------+-------------------------------+----------------------+-----------------+
|    1 | Hungary                       | HG156                | Passenger       |
|    2 | Zimbabwe                      | ZB228                | Passenger       |
|    3 | Zimbabwe                      | ZB236                | Sanitary        |
+------+-------------------------------+----------------------+-----------------+
PS C:\Users\Evil\PycharmProjects\LR#5> python .\click_ind.py select clk_data.json --type="Sanitary"
+------+-------------------------------+----------------------+-----------------+
|  No  |        Пункт назначения       |     Номер рейса      |  Тип самолета   |
+------+-------------------------------+----------------------+-----------------+
|    1 | Zimbabwe                      | ZB236                | Sanitary        |
+------+-------------------------------+----------------------+-----------------+
PS C:\Users\Evil\PycharmProjects\LR#5>
```

clk_data.json



```
[
    {
        "flight_destination": "Hungary",
        "flight_number": "HG156",
        "airplane_type": "Passenger"
    },
    {
        "flight_destination": "Zimbabwe",
        "flight_number": "ZB228",
        "airplane_type": "Passenger"
    },
    {
        "flight_destination": "Zimbabwe",
        "flight_number": "ZB236",
        "airplane_type": "Sanitary"
    }
]
```

4. Ответы на вопросы

1. В чем отличие терминала и 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 – компьютер с клавиатурой и монитором.

2. 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.

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

Python 3 поддерживает несколько различных способов обработки аргументов командной строки. Встроенный способ – использовать модуль sys. С точки зрения имен и использования, он имеет прямое отношение к

библиотеке C (libc). Второй способ – это модуль getopt , который обрабатывает как короткие, так и длинные параметры, включая оценку значений параметров. Кроме того, существуют два других общих метода. Это модуль argparse, производный от модуля optparse, доступного до Python 2.7. Другой метод – использование модуля docopt, доступного на GitHub.

4. 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().

5. 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 .

```
# Include standard modules

import getopt, sys

# Get full command-line arguments

full_cmd_arguments = sys.argv

# Keep all but the first

argument_list = full_cmd_arguments[1:]

print(argument_list)
```

Аргументы в списке аргументов теперь можно анализировать с помощью метода getopts() . Но перед этим нам нужно сообщить getopts() о том, какие параметры допустимы. Они определены так:

short_options = "ho:v"

long_options = ["help", "output=", "verbose"]

Для метода getopt() необходимо настроить три параметра – список фактических аргументов из argv, а также допустимые короткие и длинные параметры.

Сам вызов метода хранится в инструкции try - catch , чтобы скрыть ошибки во время оценки. Исключение возникает, если обнаруживается аргумент, который не является частью списка, как определено ранее. Скрипт в Python выведет сообщение об ошибке на экран и выйдет с кодом ошибки 2.

try:

arguments, values = getopt.getopt(argument_list, short_options,

long_options)

except getopt.error as err:

Output error, and return with an error code

print(str(err))

sys.exit(2)

Наконец, аргументы с соответствующими значениями сохраняются в двух переменных с именами arguments и values. Теперь вы можете легко оценить эти переменные в своем коде. Мы можем использовать цикл for для перебора списка распознанных аргументов, одна запись за другой.

Evaluate given options

for current_argument, current_value in arguments:

if current_argument in ("-v", "--verbose"):

print("Enabling verbose mode")

elif current_argument in ("-h", "--help"):

print("Displaying help")

elif current_argument in ("-o", "--output"):

print(f"Enabling special output mode ({current_value})")

Ниже вы можете увидеть результат выполнения этого кода. Далее показано, как программа реагирует как на допустимые, так и на недопустимые программные аргументы:

$ python arguments-getopt.py -h

Displaying help

$ python arguments-getopt.py --help

Displaying help

$ python arguments-getopt.py --output=green --help -v

Enabling special output mode (green)

Displaying help

Enabling verbose mode

$ python arguments-getopt.py -verbose

option -e not recognized

Последний вызов нашей программы поначалу может показаться немного запутанным. Чтобы понять это, вам нужно знать, что сокращенные параметры (иногда также называемые флагами) могут использоваться вместе с одним тире. Это позволяет вашему инструменту легче воспринимать множество вариантов.

6. Какие особенности построение CLI с использованием модуля argparse ?

Для начала рассмотрим, что интересного предлагает argparse :

- анализ аргументов sys.argv ;
- конвертирование строковых аргументов в объекты Вашей программы и работа с ними;
- форматирование и вывод информативных подсказок.

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.

Для начала работы с argparse необходимо задать парсер:

```
import argparse

parser = argparse.ArgumentParser(description='Great Description To Be Here')
```

Далее, парсеру стоит указать, какие объекты Вы от него ждете. В частном случае, это может выглядеть так:

```
parser.add_argument('-n', action='store', dest='n', help='Simple value')
```

Если действие (action) для данного аргумента не задано, то по умолчанию он будет сохраняться (store) в namespace, причем мы также можем указать тип этого аргумента (int, boolean и тд). Если имя возвращаемого аргумента (dest) задано, его значение будет сохранено в соответствующем атрибуте namespace.

В нашем случае:

```
>>> print(parser.parse_args(['-n', '3']))

Namespace(n='3')

>>> print(parser.parse_args([]))

Namespace(n=None)

>>> print(parser.parse_args(['-a', '3']))

error: unrecognized arguments: -a 3
```

Остановимся на действиях (actions). Они могут быть следующими:

store: возвращает в пространство имен значение (после необязательного приведения типа). Как уже говорилось, store — действие по умолчанию;

store_const: в основном используется для флагов. Либо вернет Вам значение, указанное в const, либо (если ничего не указано), None.

store_true / store_false: аналог store_const , но для булевых True и False ;

append: возвращает список путем добавления в него значений агрументов.

append_const: возвращение значения, определенного в спецификации аргумента, в список.

count: как следует из названия, считает, сколько раз встречается значение данного аргумента.

В зависимости от переданного в конструктор парсера аргумента add_help (булевого типа), будет определяться, включать или не включать в стандартный вывод по ключам ['-h', '--help'] сообщения о помощи. То же самое будет иметь место с аргументом version (строкового типа), ключи по умолчанию: ['-v', '--version']. При запросе помощи или номера версии, дальнейшее выполнение прерывается.

parser = argparse.ArgumentParser(add_help=True, version='4.0')