

**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 №2.17 по дисциплине основы
программной инженерии**

Выполнил:

Кожухов Филипп Денисович,
2 курс, группа ПИЖ-б-о-20-1,

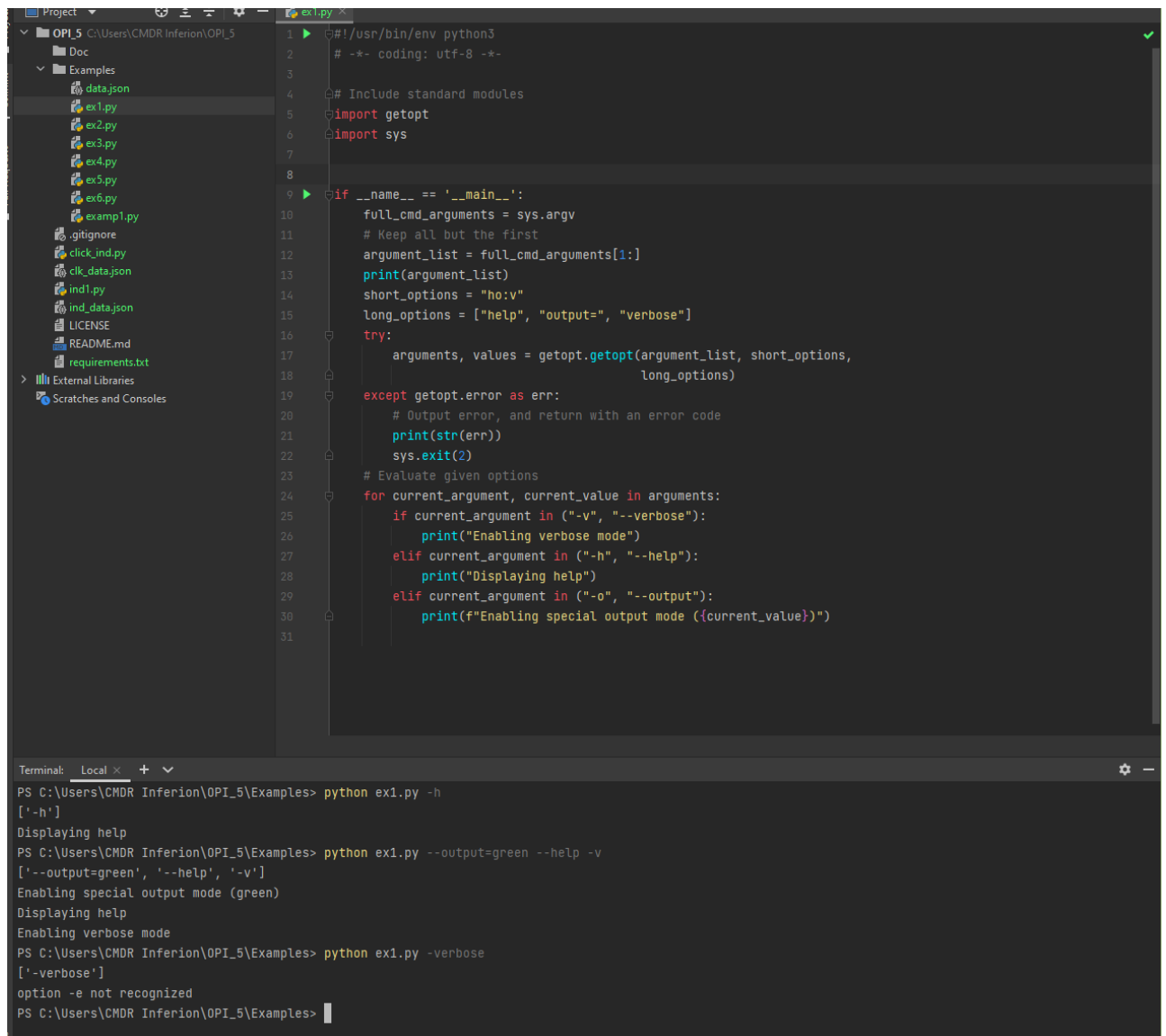
Проверил:

Доцент кафедры
прикладной математики и
компьютерной
безопасности, Воронкин Р.А.

Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2022 г.

ВЫПОЛНЕНИЕ:



The image shows a code editor with a file explorer on the left and a terminal at the bottom. The file explorer shows a project named 'OPI_5' with a subdirectory 'Examples' containing several files, including 'ex1.py'. The code editor displays the contents of 'ex1.py', which is a Python script using the 'getopt' module to parse command-line arguments. The script defines short options '-h' for help and '-v' for verbose mode, and long options '--help', '--output', and '--verbose'. It prints the list of arguments and the values of the options. The terminal shows the execution of the script with various command-line arguments, demonstrating its functionality.

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  # Include standard modules
5  import getopt
6  import sys
7
8
9  if __name__ == '__main__':
10     full_cmd_arguments = sys.argv
11     # Keep all but the first
12     argument_list = full_cmd_arguments[1:]
13     print(argument_list)
14     short_options = "ho:v"
15     long_options = ["help", "output=", "verbose"]
16     try:
17         arguments, values = getopt.getopt(argument_list, short_options,
18                                           long_options)
19     except getopt.error as err:
20         # Output error, and return with an error code
21         print(str(err))
22         sys.exit(2)
23     # Evaluate given options
24     for current_argument, current_value in arguments:
25         if current_argument in ("-v", "--verbose"):
26             print("Enabling verbose mode")
27         elif current_argument in ("-h", "--help"):
28             print("Displaying help")
29         elif current_argument in ("-o", "--output"):
30             print(f"Enabling special output mode ({current_value})")
31
```

Terminal: Local x + v

```
PS C:\Users\CMDR Inferion\OPI_5\Examples> python ex1.py -h
['-h']
Displaying help
PS C:\Users\CMDR Inferion\OPI_5\Examples> python ex1.py --output=green --help -v
['--output=green', '--help', '-v']
Enabling special output mode (green)
Displaying help
Enabling verbose mode
PS C:\Users\CMDR Inferion\OPI_5\Examples> python ex1.py -verbose
['-verbose']
option -e not recognized
PS C:\Users\CMDR Inferion\OPI_5\Examples>
```

Рисунок 1 - Пример №1

Рисунок 2 - Пример №2

```
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4 import argparse
5
6
7 if __name__ == '__main__':
8     parser = argparse.ArgumentParser()
9     parser.add_argument(
10         "square", type=int, help="display a square of a given number"
11     )
12     parser.add_argument(
13         "-v", "--verbose", action="store_true",
14         help="increase output verbosity"
15     )
16     args = parser.parse_args()
17     answer = args.square ** 2
18     if args.verbose:
19         print("the square of {} equals {}".format(args.square, answer))
20     else:
21         print(answer)
```

Terminal: Local x + v

```
PS C:\Users\CMDR Inferion\OPI_5\Examples> python ex2.py 3 27
usage: ex2.py [-h] [-v] square
ex2.py: error: unrecognized arguments: 27
PS C:\Users\CMDR Inferion\OPI_5\Examples> python ex2.py 4 6 14
usage: ex2.py [-h] [-v] square
ex2.py: error: unrecognized arguments: 6 14
PS C:\Users\CMDR Inferion\OPI_5\Examples> python ex2.py 12
144
PS C:\Users\CMDR Inferion\OPI_5\Examples> python ex2.py 9 -v
the square of 9 equals 81
PS C:\Users\CMDR Inferion\OPI_5\Examples>
```

Рисунок 3 - Пример №3

```
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4 import argparse
5
6
7 if __name__ == '__main__':
8     parser = argparse.ArgumentParser()
9     parser.add_argument("echo")
10     args = parser.parse_args()
11     print(args.echo)
```

Terminal: Local x + v

```
PS C:\Users\CMDR Inferion\OPI_5\Examples> python ex3.py hello
hello
PS C:\Users\CMDR Inferion\OPI_5\Examples> python ex3.py 78 18
usage: ex3.py [-h] echo
ex3.py: error: unrecognized arguments: 18
PS C:\Users\CMDR Inferion\OPI_5\Examples>
```

Рисунок 4 - Пример №4

Рисунок 5 - Пример №5

The screenshot shows a code editor with a file explorer on the left and a terminal at the bottom. The file explorer shows a project named 'OPI_5' with a subdirectory 'Examples' containing files like 'data.json', 'ex1.py', 'ex2.py', 'ex3.py', 'ex4.py', 'ex5.py', 'ex6.py', 'examp1.py', '.gitignore', 'click_ind.py', 'clk_data.json', 'ind1.py', 'ind_data.json', 'LICENSE', 'README.md', and 'requirements.txt'. The code editor displays a Python script for 'ex4.py' that uses 'argparse' to handle command-line arguments. It defines a 'square' argument and a '--verbosity' flag. The script calculates the square of the input number and prints the result with different verbosity levels. The terminal shows the execution of the script with various arguments: 'python ex4.py 6' (output: 36), 'python ex4.py 12 -v 2' (output: the square of 12 equals 144), and 'python ex4.py 14 -v 1' (output: 14^2 == 196).

```
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4 import argparse
5
6
7 if __name__ == '__main__':
8     parser = argparse.ArgumentParser()
9     parser.add_argument(
10         "square",
11         type=int,
12         help="display a square of a given number"
13     )
14     parser.add_argument(
15         "-v",
16         "--verbosity",
17         type=int,
18         help="increase output verbosity"
19     )
20     args = parser.parse_args()
21     answer = args.square ** 2
22     if args.verbosity == 2:
23         print("the square of {} equals {}".format(args.square, answer))
24     elif args.verbosity == 1:
25         print("{}^2 == {}".format(args.square, answer))
26     else:
27         print(answer)
```

Terminal: Local x + v

PS C:\Users\CMDR Inferion\OPI_5\Examples> python ex4.py 6
36

PS C:\Users\CMDR Inferion\OPI_5\Examples> python ex4.py 12 -v 2
the square of 12 equals 144

PS C:\Users\CMDR Inferion\OPI_5\Examples> python ex4.py 14 -v 1
14^2 == 196

PS C:\Users\CMDR Inferion\OPI_5\Examples>

Рисунок 6 - Пример №6

The screenshot shows a code editor with a file explorer on the left and a terminal at the bottom. The file explorer shows a project named 'OPI_5' with a subdirectory 'Examples' containing files like 'data.json', 'ex1.py', 'ex2.py', 'ex3.py', 'ex4.py', 'ex5.py', 'ex6.py', 'examp1.py', '.gitignore', 'click_ind.py', 'clk_data.json', 'ind1.py', 'ind_data.json', 'LICENSE', 'README.md', and 'requirements.txt'. The code editor displays a Python script for 'ex5.py' that uses 'argparse' to handle command-line arguments. It defines a 'square' argument and a '--verbosity' flag. The script calculates the square of the input number and prints the result with different verbosity levels. The terminal shows the execution of the script with various arguments: 'python ex5.py 5 -v' (output: 5^2 == 25), 'python ex5.py 8 -vv' (output: the square of 8 equals 64), and 'python ex5.py 8 -vv' (output: the square of 8 equals 64).

```
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4 import argparse
5
6
7 if __name__ == '__main__':
8     parser = argparse.ArgumentParser()
9     parser.add_argument(
10         "square",
11         type=int,
12         help="display the square of a given number"
13     )
14     parser.add_argument(
15         "-v",
16         "--verbosity",
17         action="count",
18         help="increase output verbosity"
19     )
20     args = parser.parse_args()
21     answer = args.square ** 2
22     if args.verbosity == 2:
23         print("the square of {} equals {}".format(args.square, answer))
24     elif args.verbosity == 1:
25         print("{}^2 == {}".format(args.square, answer))
26     else:
27         print(answer)
```

Terminal: Local x + v

PS C:\Users\CMDR Inferion\OPI_5\Examples> python ex5.py 5 -v
5^2 == 25

PS C:\Users\CMDR Inferion\OPI_5\Examples> python ex5.py 8 -vv
the square of 8 equals 64

PS C:\Users\CMDR Inferion\OPI_5\Examples>

The image shows a code editor with a file explorer on the left and a terminal at the bottom. The file explorer displays a project named 'OPI_5' with a subdirectory 'Examples' containing several files, including 'ex6.py' which is currently selected. The code editor shows the content of 'ex6.py', a Python script that uses the 'argparse' module to parse command-line arguments. The script defines two arguments: 'x' (an integer representing a base) and 'y' (an integer representing an exponent). It also includes a verbose flag '-v' that counts the number of times it is used. The script calculates the power of x to the y and prints the result, with additional formatting for verbose output. The terminal at the bottom shows the execution of the script with two different sets of arguments: 'python ex6.py 4 6' and 'python ex6.py 2 15', resulting in the outputs '4096' and '32768' respectively.

```
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4 import argparse
5
6
7 if __name__ == '__main__':
8     parser = argparse.ArgumentParser()
9     parser.add_argument("x", type=int, help="the base")
10    parser.add_argument("y", type=int, help="the exponent")
11    parser.add_argument("-v", "--verbosity", action="count", default=0)
12    args = parser.parse_args()
13    answer = args.x ** args.y
14    if args.verbosity >= 2:
15        print("{} to the power {} equals {}".format(args.x, args.y, answer))
16    elif args.verbosity >= 1:
17        print("{}^{} == {}".format(args.x, args.y, answer))
18    else:
19        print(answer)
20
```

Terminal: Local × + ▾

```
PS C:\Users\CMDR Inferion\OPI_5\Examples> python ex6.py 4 6
4096
PS C:\Users\CMDR Inferion\OPI_5\Examples> python ex6.py 2 15
32768
PS C:\Users\CMDR Inferion\OPI_5\Examples>
```

```

1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  import argparse
5  import json
6  import os.path
7  from datetime import date
8
9
10 def add_worker(staff, name, post, year):
11     """
12     Добавить данные о работнике.
13     """
14     staff.append(
15         {
16             "name": name,
17             "post": post,
18             "year": year
19         }
20     )
21     return staff
22
23
24 def display_workers(staff):
25     """
26     Отобразить список работников.
27     """
28     # Проверить, что список работников не пуст.
29     if staff:
30         # Заголовок таблицы.
31         line = '+-{}-+-{}-+-{}-+-+'.format(
32             '-' * 4,
33             '-' * 30,
34             '-' * 20,
35             '-' * 8
36         )
37         print(line)
38         print(
39             '| {:^4} | {:^30} | {:^20} | {:^8} |'.format(
40                 "No",
41                 "Ф.И.О.",
42                 "Должность",
43                 "Год"
44             )
45         )
46         print(line)
47
48         # Вывести данные о всех сотрудниках.
49         for idx, worker in enumerate(staff, 1):
50             print(
51                 '| {:>4} | {:<30} | {:<20} | {:>8} |'.format(
52                     idx,
53                     worker.get('name', ''),
54                     worker.get('post', ''),
55                     worker.get('year', 0)
56                 )
57             )
58             print(line)
59     else:
60         print("Список работников пуст.")
61
62
63 def select_workers(staff, period):
64     """
65     Выбрать работников с заданным стажем.
66     """
67     # Получить текущую дату.
68     today = date.today()
69     # Сформировать список работников.
70     result = []
71     for employee in staff:
72         if today.year - employee.get('year', today.year) >= period:
73             result.append(employee)
74     # Возвратить список выбранных работников.
75     return result
76
77
78 def save_workers(file_name, staff):
79     """
80     Сохранить всех работников в файл JSON.
81     """
82     # Открыть файл с заданным именем для записи.
83     with open(file_name, "w", encoding="utf-8") as fout:
84         # Выполнить сериализацию данных в формат JSON.
85         # Для поддержки кириллицы установим ensure_ascii=False
86         json.dump(staff, fout, ensure_ascii=False, indent=4)
87
88
89 def load_workers(file_name):
90     """
91     Загрузить всех работников из файла JSON.
92     """
93     # Открыть файл с заданным именем для чтения.
94     with open(file_name, "r", encoding="utf-8") as fin:
95         return json.load(fin)
96

```

Рисунки
Пример
Рисунок
Вывод

```
98 def main(command_line=None):
99     # Создать родительский парсер для определения имени файла.
100     file_parser = argparse.ArgumentParser(add_help=False)
101     file_parser.add_argument(
102         "filename",
103         action="store",
104         help="The data file name"
105     )
106     # Создать основной парсер командной строки.
107     parser = argparse.ArgumentParser("workers")
108     parser.add_argument(
109         "--version",
110         action="version",
111         version="% (prog)s 0.1.0"
112     )
113     subparsers = parser.add_subparsers(dest="command")
114     # Создать субпарсер для добавления работника.
115     add = subparsers.add_parser(
116         "add",
117         parents=[file_parser],
118         help="Add a new worker"
119     )
120     add.add_argument(
121         "-n",
122         "--name",
123         action="store",
124         required=True,
125         help="The worker's name"
126     )
127     add.add_argument(
128         "-p",
129         "--post",
130         action="store",
131         help="The worker's post"
132     )
133     add.add_argument(
134         "-y",
135         "--year",
136         action="store",
137         type=int,
138         required=True,
139         help="The year of hiring"
140     )
141     # Создать субпарсер для отображения всех работников.
142     _ = subparsers.add_parser(
143         "display",
144         parents=[file_parser],
145         help="Display all workers"
146     )
147     # Создать субпарсер для выбора работников.
148     select = subparsers.add_parser(
149         "select",
150         parents=[file_parser],
151         help="Select the workers"
152     )
153     select.add_argument(
154         "-p",
155         "--period",
156         action="store",
157         type=int,
158         required=True,
159         help="The required period"
160     )
161     # Выполнить разбор аргументов командной строки.
162     args = parser.parse_args(command_line)
163     # Загрузить всех работников из файла, если файл существует.
164     is_dirty = False
165     if os.path.exists(args.filename):
166         workers = load_workers(args.filename)
167     else:
168         workers = []
169     # Добавить работника.
170     if args.command == "add":
171         workers = add_worker(
172             workers,
173             args.name,
174             args.post,
175             args.year
176         )
177     is_dirty = True
178     # Отобразить всех работников.
179     elif args.command == "display":
180         display_workers(workers)
181     # Выбрать требуемых работников.
182     elif args.command == "select":
183         selected = select_workers(workers, args.period)
184         display_workers(selected)
185     # Сохранить данные в файл, если список работников был изменен.
186     if is_dirty:
187         save_workers(args.filename, workers)
188
189
190 if __name__ == "__main__":
191     main()
192
```

7-10 -
№7
11 -

программы

```
Terminal: Local x + -
PS C:\Users\CMDR Inferion\OPI_5\Examples> python ex7.py add data.json --name="Cdqcqs Wehmn" --post="ASdkhjcqjsk" --year="2016"
PS C:\Users\CMDR Inferion\OPI_5\Examples> python ex7.py add data.json --name="Ldkk Ebmds" --post="SSNnq" --year="2018"
PS C:\Users\CMDR Inferion\OPI_5\Examples> python ex7.py add data.json --name="JmkLe Ksnhj" --post="Mjsdh" --year="2009"
PS C:\Users\CMDR Inferion\OPI_5\Examples> python ex7.py display data.json

+-----+-----+-----+-----+
| No |          Ф.И.О.          |      Должность      |      Год      |
+-----+-----+-----+-----+
| 1 | Cdqcqs Wehmn          | ASdkhjcqjsk        | 2016 |
| 2 | Ldkk Ebmds           | SSNnq              | 2018 |
| 3 | JmkLe Ksnhj          | Mjsdh              | 2009 |
+-----+-----+-----+-----+

PS C:\Users\CMDR Inferion\OPI_5\Examples> python ex7.py select data.json --period=5

+-----+-----+-----+-----+
| No |          Ф.И.О.          |      Должность      |      Год      |
+-----+-----+-----+-----+
| 1 | Cdqcqs Wehmn          | ASdkhjcqjsk        | 2016 |
| 2 | JmkLe Ksnhj          | Mjsdh              | 2009 |
+-----+-----+-----+-----+

PS C:\Users\CMDR Inferion\OPI_5\Examples>
```

ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ


```

1  #!/usr/bin/env python3
2  #-*- coding: utf-8 -*-
3
4  import argparse
5  import json
6  import os.path
7
8
9  def get_flight(fls, dest, num, type):
10     """
11     Добавить данные в работники.
12     """
13     fls.append(
14         {
15             "flight_destination": dest,
16             "flight_number": num,
17             "airplane_type": type
18         }
19     )
20     return fls
21
22
23 def display_flights(flights):
24     """
25     Отобразить список рейсов
26     """
27     if flights:
28         line = '+--+--+--+--+--+'.format(
29             '-' * 4,
30             '-' * 30,
31             '-' * 20,
32             '-' * 15
33         )
34         print(line)
35         print(
36             '| {:^4} | {:^30} | {:^20} | {:^15} |'.format(
37                 "No",
38                 "Пункт назначения",
39                 "Номер рейса",
40                 "Тип самолета"
41             )
42         )
43         print(line)
44         for idx, flight in enumerate(flights, 1):
45             print(
46                 '| {:^4} | {:^30} | {:^20} | {:^15} |'.format(
47                     idx,
48                     flight.get('flight_destination', ''),
49                     flight.get('flight_number', ''),
50                     flight.get('airplane_type', 0)
51                 )
52             )
53             print(line)
54     else:
55         print("Список рейсов пуст")
56
57
58 def select_flights(flights, airplane_type):
59     """
60     Выбрать рейсы самолётов заданного типа
61     """
62     count = 0
63     res = []
64     for flight in flights:
65         if flight.get('airplane_type') == airplane_type:
66             count += 1
67             res.append(flight)
68     if count == 0:
69         print("рейсы не найдены")
70     return res
71
72
73 def save_flights(file_name, fls):
74     """
75     Сохранить все записи полётов в файл JSON.
76     """
77     # Открыть файл с заданным именем для записи.
78     with open(file_name, "w", encoding="utf-8") as fout:
79         # Выполнить сериализацию данных в формат JSON.
80         # Для поддержки кириллицы установим ensure_ascii=False
81         json.dump(fls, fout, ensure_ascii=False, indent=4)
82
83
84 def load_flights(file_name):
85     """
86     Загрузить все записи полётов из файла JSON.
87     """
88     # Открыть файл с заданным именем для чтения.
89     with open(file_name, "r", encoding="utf-8") as fin:
90         return json.load(fin)
91
92
93 def main(command_line=None):
94     """
95     Главная функция программы
96     """
97     file_parser = argparse.ArgumentParser(add_help=False)
98     file_parser.add_argument(
99         "filename",
100         action="store",
101         help="The data file name"
102     )
103     parser = argparse.ArgumentParser("flights")
104     parser.add_argument(
105         "--version",
106         action="version",
107         version="%prog) 0.1.0"
108     )
109

```

```

111     subparsers = parser.add_subparsers(dest="command")
112     add = subparsers.add_parser(
113         "add",
114         parents=[file_parser],
115         help="Add a new flight"
116     )
117     add.add_argument(
118         "-fld",
119         "--flight_dest",
120         action="store",
121         required=True,
122         help="The flight destination"
123     )
124     add.add_argument(
125         "-n",
126         "--number",
127         action="store",
128         help="The flight number"
129     )
130     add.add_argument(
131         "-t",
132         "--type",
133         action="store",
134         required=True,
135         help="The airplane type"
136     )
137     _ = subparsers.add_parser(
138         "display",
139         parents=[file_parser],
140         help="Display all flights"
141     )
142     select = subparsers.add_parser(
143         "select",
144         parents=[file_parser],
145         help="Select the flights"
146     )
147     select.add_argument(
148         "-t",
149         "--type",
150         action="store",
151         required=True,
152         help="The required flight type"
153     )
154     args = parser.parse_args(command_line)
155     is_dirty = False
156     if os.path.exists(args.filename):
157         flights = load_flights(args.filename)
158     else:
159         flights = []
160     if args.command == "add":
161         flights = get_flight(
162             flights,
163             args.flight_dest,
164             args.number,
165             args.type
166         )
167     is_dirty = True
168     elif args.command == "display":
169         display_flights(flights)
170     elif args.command == "select":
171         selected = select_flights(flights, args.type)
172         display_flights(selected)
173     if is_dirty:
174         save_flights(args.filename, flights)
175
176
177 if __name__ == '__main__':
178     main()
179

```

Рисунки 12-13 - Код ИДЗ

Рисунок 14 - Вывод программы

```

PS C:\Users\CMDR Inferion\OPI_5> python individual.py add individual_data.json -fld="Japan" -n="RF213" -t="Cargo"
PS C:\Users\CMDR Inferion\OPI_5> python individual.py add individual_data.json -fld="Uganda" -n="JQ412" -t="Military"
PS C:\Users\CMDR Inferion\OPI_5> python individual.py display individual_data.json
+-----+-----+-----+
| No | Пункт назначения | Номер рейса | Тип самолета |
+-----+-----+-----+
| 1 | Thailand | TH130 | Passenger |
| 2 | Moscow | M0234 | Military |
| 3 | Japan | RF213 | Cargo |
| 4 | Uganda | JQ412 | Military |
+-----+-----+-----+
PS C:\Users\CMDR Inferion\OPI_5> python individual.py select individual_data.json --type="Cargo"
+-----+-----+-----+
| No | Пункт назначения | Номер рейса | Тип самолета |
+-----+-----+-----+
| 1 | Japan | RF213 | Cargo |
+-----+-----+-----+
PS C:\Users\CMDR Inferion\OPI_5>

```

ЗАДАНИЕ ПОВЫШЕННОЙ СЛОЖНОСТИ

```

1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4 import click
5 import json
6 import os.path
7
8
9 def get_flight(fls, dest, num, type):
10     """
11     Добавить данные о работнике.
12     """
13     fls.append(
14         {
15             "flight_destination": dest,
16             "flight_number": num,
17             "airplane_type": type
18         }
19     )
20     return fls
21
22
23 def display_flights(flights):
24     """
25     Отобразить список рейсов
26     """
27     if flights:
28         line = '{}{}{}{}{}{}'.format(
29             '-' * 4,
30             '-' * 30,
31             '-' * 20,
32             '-' * 15
33         )
34         print(line)
35         print(
36             '| {:^4} | {:^30} | {:^20} | {:^15} |'.format(
37                 "No",
38                 "Пункт назначения",
39                 "Номер рейса",
40                 "Тип самолета"
41             )
42         )
43         print(line)
44         for idx, flight in enumerate(flights, 1):
45             print(
46                 '| {:>4} | {:<30} | {:<20} | {:<15} |'.format(
47                     idx,
48                     flight.get('flight_destination', ''),
49                     flight.get('flight_number', ''),
50                     flight.get('airplane_type', '')
51                 )
52             )
53             print(line)
54     else:
55         print("Список рейсов пуст")
56
57
58 def select_flights(flights, airplane_type):
59     """
60     Выбрать рейсы самолетов заданного типа
61     """
62     count = 0
63     res = []
64     for flight in flights:
65         if flight.get('airplane_type') == airplane_type:
66             count += 1
67             res.append(flight)
68     if count == 0:
69         print("рейсы не найдены")
70     return res
71
72
73 def save_flights(file_name, fls):
74     """
75     Сохранить все записи полётов в файл JSON.
76     """
77     # Открыть файл с заданным именем для записи.
78     with open(file_name, "w", encoding="utf-8") as fout:
79         # Выполнить сериализацию данных в формат JSON.
80         # Для поддержки кириллицы установим ensure_ascii=False
81         json.dump(fls, fout, ensure_ascii=False, indent=4)
82
83
84 def load_flights(file_name):
85     """
86     Загрузить все записи полётов из файла JSON.
87     """
88     # Открыть файл с заданным именем для чтения.
89     with open(file_name, "r", encoding="utf-8") as fin:
90         return json.load(fin)
91
92
93 @click.Command()
94 @click.argument('command')
95 @click.argument('filename')
96 @click.option('--flight_dest', prompt='Destination?',
97               help='The flight destination')
98 @click.option('--number', prompt='Enter flight num', help='The flight number')
99 @click.option('--type', prompt='Enter airplane type', help='The airplane type')
100
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)

```

Рисунки

Рисунок

15-17 - Код
программы

18 - Вывод программы

```

110         else:
111             flights = []
112         if command == "add":
113             flights = get_flight(
114                 flights,
115                 flight_dest,
116                 number,
117                 type
118             )
119             is_dirty = True
120         elif command == "display":
121             display_flights(flights)
122         elif command == "select":
123             selected = select_flights(flights, type)
124             display_flights(selected)
125         if is_dirty:
126             save_flights(filename, flights)
127
128
129 if __name__ == '__main__':
130     main()
131

```

```
PS C:\Users\CMDR Inferion\OPI_5> python .\CLICK.py add click_data.json --flight_dest="Paraguay" --number="SDQ14" --type="Cargo"
```

```
PS C:\Users\CMDR Inferion\OPI_5> python .\CLICK.py display click_data.json
```

```
Destination?: Paraguay
```

```
Enter flight num: SDQ14
```

```
Enter airplane type: Cargo
```

No	Пункт назначения	Номер рейса	Тип самолета
1	Hungary	H6156	Passenger
2	Zimbabwe	ZB228	Passenger
3	Zimbabwe	ZB236	Sanitary
4	Paraguay	SDQ14	Cargo
5	Paraguay	SDQ14	Cargo

```
PS C:\Users\CMDR Inferion\OPI_5> python .\CLICK.py select click_data.json --type="Cargo"
```

```
Destination?: Paraguay
```

```
Enter flight num: SDQ14
```

No	Пункт назначения	Номер рейса	Тип самолета
1	Paraguay	SDQ14	Cargo
2	Paraguay	SDQ14	Cargo

```
PS C:\Users\CMDR Inferion\OPI_5>
```

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 фасы. 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 типе. 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 отправляет 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

аргументов.

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')
```