

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

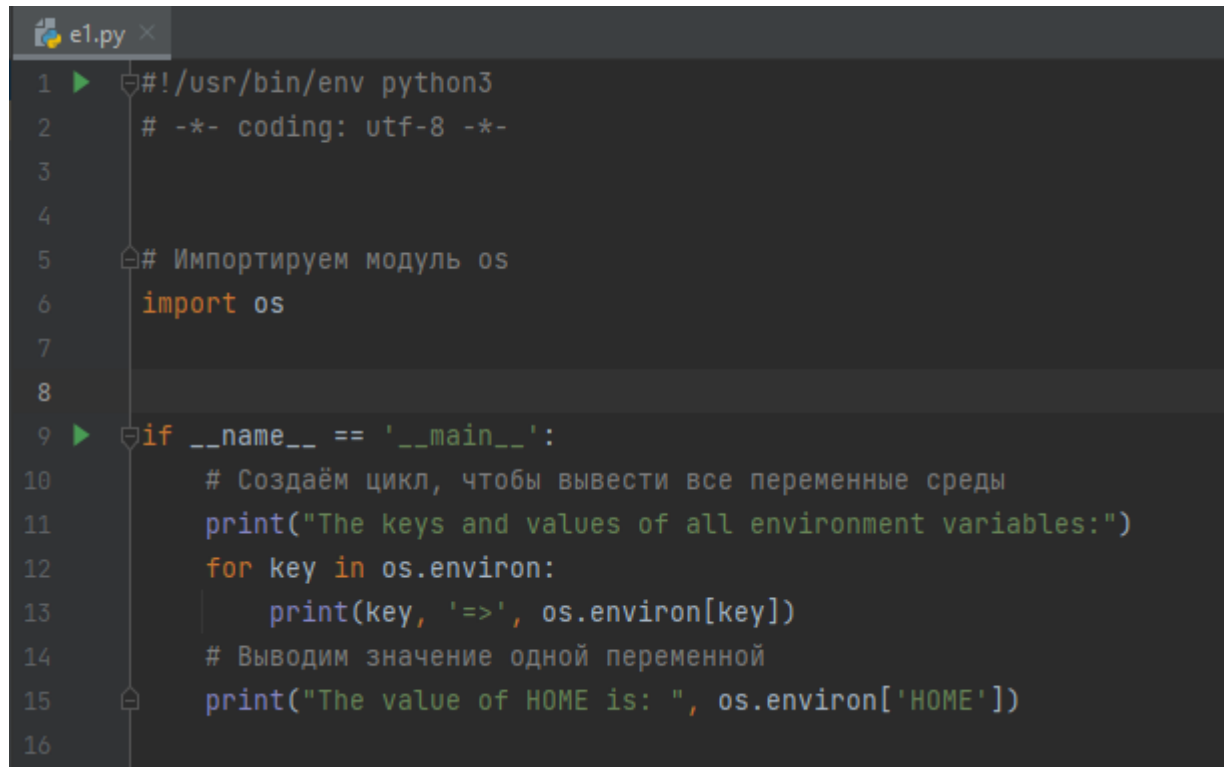
**Отчет о лабораторной работе №6 по дисциплине основы программной
инженерии**

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

A screenshot of a code editor window titled 'e1.py'. The editor shows a Python script with 16 lines of code. The code is written in a dark-themed editor with syntax highlighting. The script starts with a shebang line and a UTF-8 encoding declaration. It then imports the 'os' module. A main block is defined using an 'if __name__ == '__main__':' guard. Inside this block, there are three print statements: one to print all environment variables, one to iterate over them and print each key-value pair, and one to print the value of the 'HOME' variable specifically. The code is as follows:

```
1  ▶  #!/usr/bin/env python3
2      # -*- coding: utf-8 -*-
3
4
5  ⊞  # Импортируем модуль os
6      import os
7
8
9  ▶  if __name__ == '__main__':
10     # Создаём цикл, чтобы вывести все переменные среды
11     print("The keys and values of all environment variables:")
12     for key in os.environ:
13         print(key, '=>', os.environ[key])
14     # Выводим значение одной переменной
15     print("The value of HOME is:", os.environ['HOME'])
16
```

```
e1 (1) x
IDEA_INITIAL_DIRECTORY => C:\Users\Evil\Desktop
LOCALAPPDATA => C:\Users\Evil\AppData\Local
LOGONSERVER => \\WIN-RH2MAR4K060
NUMBER_OF_PROCESSORS => 8
ONEDRIVE => C:\Users\Evil\OneDrive
OS => Windows_NT
PATH => C:\Users\Evil\PycharmProjects\LR #6\venv\Scripts;C:\Program Files (x86)\Common File
PATHEXT => .COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
PROCESSOR_ARCHITECTURE => AMD64
PROCESSOR_IDENTIFIER => AMD64 Family 21 Model 2 Stepping 0, AuthenticAMD
PROCESSOR_LEVEL => 21
PROCESSOR_REVISION => 0200
PROGRAMDATA => C:\ProgramData
PROGRAMFILES => C:\Program Files
PROGRAMFILES(X86) => C:\Program Files (x86)
PROGRAMW6432 => C:\Program Files
PROMPT => (venv) $P$G
PSMODULEPATH => C:\Program Files\WindowsPowerShell\Modules;C:\WINDOWS\system32\WindowsPower
PUBLIC => C:\Users\Public
PYCHARM_HOSTED => 1
PYTHONIOENCODING => UTF-8
PYTHONPATH => C:\Users\Evil\PycharmProjects\LR #6
PYTHONUNBUFFERED => 1
SESSIONNAME => Console
SYSTEMDRIVE => C:
SYSTEMROOT => C:\WINDOWS
TEMP => C:\Users\Evil\AppData\Local\Temp
TMP => C:\Users\Evil\AppData\Local\Temp
USERDOMAIN => WIN-RH2MAR4K060
USERDOMAIN_ROAMINGPROFILE => WIN-RH2MAR4K060
USERNAME => Evil
USERPROFILE => C:\Users\Evil
VBOX_MSI_INSTALL_PATH => C:\Program Files\Oracle\VirtualBox\
VIRTUAL_ENV => C:\Users\Evil\PycharmProjects\LR #6\venv
WINDIR => C:\WINDOWS
_OLD_VIRTUAL_PATH => C:\Program Files (x86)\Common Files\Oracle\Java\javapath;C:\Windows\sy
_OLD_VIRTUAL_PROMPT => $P$G

Process finished with exit code 1
```

```
e1.py x e2.py x
6 import sys
7
8
9 if __name__ == '__main__':
10     while True:
11         # Принимаем имя переменной среды
12         key_value = input("Enter the key of the environment variable:")
13         # Проверяем, инициализирована ли переменная
14         try:
15             if os.environ[key_value]:
16                 print(
17                     "The value of",
18                     key_value,
19                     " is ",
20                     os.environ[key_value]
21                 )
22             # Если переменной не присвоено значение, то ошибка
23         except KeyError:
24             print(key_value, 'environment variable is not set.')
25             # Завершаем процесс выполнения скрипта
26             sys.exit(1)
27
```

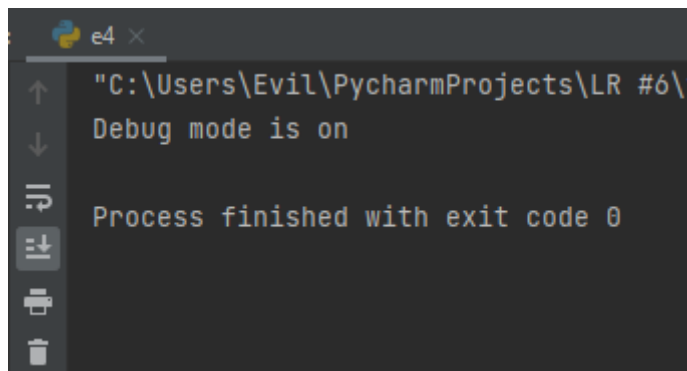
```
e2 x
"C:\Users\Evil\PycharmProjects\LR #6\venv\Scripts\python.exe"
Enter the key of the environment variable:USERNAME
The value of USERNAME is Evil
Enter the key of the environment variable:PATH
The value of PATH is C:\Users\Evil\PycharmProjects\LR #6\venv\Scripts\python.exe
Enter the key of the environment variable:HOME
HOME environment variable is not set.

Process finished with exit code 1
```

```
e1.py x e2.py x e3.py x
1 ▶ #!/usr/bin/env python3
2   # -*- coding: utf-8 -*-
3
4   # Импортируем модуль os
5   import os
6
7
8 ▶ if __name__ == '__main__':
9     # Проверяем значение переменной среды
10    if os.environ.get('DEBUG') == 'True':
11        print('Debug mode is on')
12    else:
13        print('Debug mode is off')
14
```

```
e3 x
↑ "C:\Users\Evil\PycharmProjects\LR #
↓ Debug mode is off
⌵ Process finished with exit code 0
⌵
⌵
⌵
```

```
e1.py x e2.py x e3.py x e4.py x
1 ▶ #!/usr/bin/env python3
2   # -*- coding: utf-8 -*-
3
4   # Импортируем модуль os
5   import os
6
7
8 ▶ if __name__ == '__main__':
9     # Задаём значение переменной DEBUG
10    os.environ.setdefault('DEBUG', 'True')
11    # Проверяем значение переменной среды
12    if os.environ.get('DEBUG') == 'True':
13        print('Debug mode is on')
14    else:
15        print('Debug mode is off')
16
```



The image shows a PyCharm console window with a dark theme. The title bar at the top indicates the file 'e4' is open. The console output consists of two lines: the first line shows the file path 'C:\Users\Evil\PycharmProjects\LR #6\' followed by the text 'Debug mode is on'; the second line states 'Process finished with exit code 0'. On the left side of the console, there is a vertical toolbar with icons for navigating through the output (up, down, first, last, previous, next) and a trash icon at the bottom.

```
e4 x
"C:\Users\Evil\PycharmProjects\LR #6\
Debug mode is on
Process finished with exit code 0
```

```
ex5.py x
1  ▶  #!/usr/bin/env python3
2      # -*- coding: utf-8 -*-
3
4
5      import argparse
6      import json
7      import os
8      import sys
9      from datetime import date
10
11
12      def add_worker(staff, name, post, year):
13          """
14          Добавить данные о работнике.
15          """
16          staff.append(
17              {
18                  "name": name,
19                  "post": post,
20                  "year": year
21              }
22          )
23          return staff
24
25
26      def display_workers(staff):
27          """
28          Отобразить список работников.
29          """
30          # Проверить, что список работников не пуст.
31          if staff:
32              # Заголовок таблицы.
33              line = '+-{}-+-{}-+-{}-+-{}-+'.format(
34                  '-' * 4,
35                  '-' * 30,
36                  '-' * 20,
37                  '-' * 8
38              )
39              print(line)
```

```
40         print(
41             '| {:^4} | {:^30} | {:^20} | {:^8} |'.format(
42                 "No",
43                 "Ф.И.О.",
44                 "Должность",
45                 "Год"
46             )
47         )
48     print(line)
49     # Вывести данные о всех сотрудниках.
50
51     for idx, worker in enumerate(staff, 1):
52         print(
53             '| {:>4} | {:<30} | {:<20} | {:>8} |'.format(
54                 idx,
55                 worker.get('name', ''),
56                 worker.get('post', ''),
57                 worker.get('year', 0)
58             )
59         )
60     print(line)
61 else:
62     print("Список работников пуст.")
63
64
65 def select_workers(staff, period):
66     """
67     Выбрать работников с заданным стажем.
68     """
69     # Получить текущую дату.
70     today = date.today()
71     # Сформировать список работников.
72     result = []
73     for employee in staff:
74         if today.year - employee.get('year', today.year) >= period:
75             result.append(employee)
76     # Возвратить список выбранных работников.
77     return result
78
```



```
ex5.py x
79
80 def save_workers(file_name, staff):
81     """
82     Сохранить всех работников в файл JSON.
83     """
84     # Открыть файл с заданным именем для записи.
85     with open(file_name, "w", encoding="utf-8") as fout:
86         # Выполнить сериализацию данных в формат JSON.
87         # Для поддержки кириллицы установим ensure_ascii=False
88         json.dump(staff, fout, ensure_ascii=False, indent=4)
89
90
91 def load_workers(file_name):
92     """
93     Загрузить всех работников из файла JSON.
94     """
95     # Открыть файл с заданным именем для чтения.
96     with open(file_name, "r", encoding="utf-8") as fin:
97         return json.load(fin)
98
99
100 def main(command_line=None):
101     # Создать родительский парсер для определения имени файла.
102     file_parser = argparse.ArgumentParser(add_help=False)
103     file_parser.add_argument(
104         "-d",
105         "--data",
106         action="store",
107         required=False,
108         help="The data file name"
109     )
110     # Создать основной парсер командной строки.
111     parser = argparse.ArgumentParser("workers")
112     parser.add_argument(
113         "--version",
114         action="version",
115         version="%(prog)s 0.1.0"
116     )
117     subparsers = parser.add_subparsers(dest="command")
```

```
118 # Создать субпарсер для добавления работника.
119 add = subparsers.add_parser(
120     "add",
121     parents=[file_parser],
122     help="Add a new worker"
123 )
124 add.add_argument(
125     "-n",
126     "--name",
127     action="store",
128     required=True,
129     help="The worker's name"
130 )
131 add.add_argument(
132     "-p",
133     "--post",
134     action="store",
135     help="The worker's post"
136 )
137 add.add_argument(
138     "-y",
139     "--year",
140     action="store",
141     type=int,
142     required=True,
143     help="The year of hiring"
144 )
145 # Создать субпарсер для отображения всех работников.
146 _ = subparsers.add_parser(
147     "display",
148     parents=[file_parser],
149     help="Display all workers"
150 )
151 # Создать субпарсер для выбора работников.
152 select = subparsers.add_parser(
153     "select",
154     parents=[file_parser],
155     help="Select the workers"
156 )
```

ex5.py x

```
157     select.add_argument(  
158         "-p",  
159         "--period",  
160         action="store",  
161         type=int,  
162         required=True,  
163         help="The required period"  
164     )  
165     # Выполнить разбор аргументов командной строки.  
166     args = parser.parse_args(command_line)  
167     # Получить имя файла.  
168     data_file = args.data  
169     os.environ.setdefault('WORKERS_DATA', 'data_file')  
170     if not data_file:  
171         data_file = os.environ.get("WORKERS_DATA")  
172     if not data_file:  
173         print("The data file name is absent", file=sys.stderr)  
174         sys.exit(1)  
175     # Загрузить всех работников из файла, если файл существует.  
176     is_dirty = False  
177     if os.path.exists(data_file):  
178         workers = load_workers(data_file)  
179     else:  
180         workers = []  
181     # Добавить работника.  
182     if args.command == "add":  
183         workers = add_worker(  
184             workers,  
185             args.name,  
186             args.post,  
187             args.year  
188         )  
189         is_dirty = True  
190     # Отобразить всех работников.  
191     elif args.command == "display":  
192         display_workers(workers)  
193     # Выбрать требуемых работников.  
194     elif args.command == "select":  
195         selected = select_workers(workers, args.period)
```

```

195         selected = select_workers(workers, args.period)
196         display_workers(selected)
197         # Сохранить данные в файл, если список работников был изменен.
198         if is_dirty:
199             save_workers(data_file, workers)
200
201
202 if __name__ == "__main__":
203     main()
204

```

```

PS C:\Users\Evil\PycharmProjects\LR #6> python ex5.py add --data="workers2.json" --name="Hdddj" --post="Bdnddj" --year=2010
workers2.json
PS C:\Users\Evil\PycharmProjects\LR #6> python ex5.py display
None
Список работников пуст.
PS C:\Users\Evil\PycharmProjects\LR #6> python ex5.py display
None
Список работников пуст.
PS C:\Users\Evil\PycharmProjects\LR #6> python ex5.py add --name="Hdj" --post="Bdj" --year=2013
None
PS C:\Users\Evil\PycharmProjects\LR #6> python ex5.py display
None
+-----+-----+-----+-----+
| No |          Ф.И.О.          |      Должность      |      Год      |
+-----+-----+-----+-----+
|  1 | Hdj                      | Bdj                  | 2013          |
+-----+-----+-----+-----+
PS C:\Users\Evil\PycharmProjects\LR #6> python ex5.py add --name="dj" --post="PBpaooaj" --year=2019
PS C:\Users\Evil\PycharmProjects\LR #6> python ex5.py display
+-----+-----+-----+-----+
| No |          Ф.И.О.          |      Должность      |      Год      |
+-----+-----+-----+-----+
|  1 | Hdj                      | Bdj                  | 2013          |
+-----+-----+-----+-----+
|  2 | dj                      | PBpaooaj             | 2019          |
+-----+-----+-----+-----+
PS C:\Users\Evil\PycharmProjects\LR #6> 

```

2. Индивидуальное задание №1 (рис 1-6).

```
idz1.py × flights.json ×
1 ▶ #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4 import argparse
5     import json
6     import os.path
7 import sys
8
9
10 def get_flight(fls, dest, num, type):
11     """
12     Добавить данные о работнике.
13     """
14     fls.append(
15         {
16             "flight_destination": dest,
17             "flight_number": num,
18             "airplane_type": type
19         }
20     )
21     return fls
22
23
24 def display_flights(flights):
25     """
26     Отобразить список рейсов
27     """
28     if flights:
29         line = '+-{}-+-{}-+-{}-+-{}-+'.format(
30             '-' * 4,
31             '-' * 30,
32             '-' * 20,
33             '-' * 15
34         )
35         print(line)
36         print(
37             '| {:^4} | {:^30} | {:^20} | {:^15} |'.format(
38                 "No",
39                 "Пункт назначения",
```

Рисунок 1 – Код программы

```
idz1.py × flights.json ×
40         "Номер рейса",
41         "Тип самолета"
42     )
43 )
44 print(line)
45 for idx, flight in enumerate(flights, 1):
46     print(
47         '| {:>4} | {:<30} | {:<20} | {:<15} |'.format(
48             idx,
49             flight.get('flight_destination', ''),
50             flight.get('flight_number', ''),
51             flight.get('airplane_type', 0)
52         )
53     )
54     print(line)
55
56 else:
57     print("Список рейсов пуст")
58
59
60 def select_flights(flights, airplane_type):
61     """
62     Выбрать рейсы самолётов заданного типа
63     """
64     count = 0
65     res = []
66     for flight in flights:
67         if flight.get('airplane_type') == airplane_type:
68             count += 1
69             res.append(flight)
70     if count == 0:
71         print("рейсы не найдены")
72
73     return res
74
75
76 def save_flights(file_name, fls):
77     """
78     Сохранить все записи полётов в файл JSON.
79
80     main() > if is_dirty
```

Рисунок 2 – Код программы, продолжение

```
idz1.py x flights.json x
79      """
80      # Открыть файл с заданным именем для записи.
81      with open(file_name, "w", encoding="utf-8") as fout:
82          # Выполнить сериализацию данных в формат JSON.
83          # Для поддержки кириллицы установим ensure_ascii=False
84          json.dump(fls, fout, ensure_ascii=False, indent=4)
85
86
87      def load_flights(file_name):
88          """
89          Загрузить все записи полётов из файла JSON.
90          """
91          # Открыть файл с заданным именем для чтения.
92          with open(file_name, "r", encoding="utf-8") as fin:
93              return json.load(fin)
94
95
96      def main(command_line=None):
97          """
98          Главная функция программы
99          """
100         # Создать родительский парсер для определения имени файла.
101         file_parser = argparse.ArgumentParser(add_help=False)
102         file_parser.add_argument(
103             "-d",
104             "--data",
105             action="store",
106             required=False,
107             help="The data file name"
108         )
109         parser = argparse.ArgumentParser("flights")
110         parser.add_argument(
111             "--version",
112             action="version",
113             version="%(prog)s 0.1.0"
114         )
115         subparsers = parser.add_subparsers(dest="command")
116         add = subparsers.add_parser(
117             "add",
118
main() > if is_dirty
```

Рисунок 3 – Код программы, продолжение

```
idz1.py × flights.json ×
118     parents=[file_parser],
119     help="Add a new flight"
120 )
121 add.add_argument(
122     "-fld",
123     "--flight_dest",
124     action="store",
125     required=True,
126     help="The flight destination"
127 )
128 add.add_argument(
129     "-n",
130     "--number",
131     action="store",
132     help="The flight number"
133 )
134 add.add_argument(
135     "-t",
136     "--type",
137     action="store",
138     required=True,
139     help="The airplane type"
140 )
141 _ = subparsers.add_parser(
142     "display",
143     parents=[file_parser],
144     help="Display all flights"
145 )
146 select = subparsers.add_parser(
147     "select",
148     parents=[file_parser],
149     help="Select the flights"
150 )
151 select.add_argument(
152     "-t",
153     "--type",
154     action="store",
155     required=True,
156     help="The required flight type"

main() > if is_dirty
```

Рисунок 4 – Код программы, продолжение


```
idz1.py x flights.json x
156         help="The required flight type"
157     )
158     args = parser.parse_args(command_line)
159     data_file = args.data
160     if not data_file:
161         os.environ.setdefault('WORKERS_DATA', 'flights.json')
162         data_file = os.environ.get("WORKERS_DATA")
163     if not data_file:
164         print("The data file name is absent", file=sys.stderr)
165         sys.exit(1)
166     is_dirty = False
167     if os.path.exists(data_file):
168         flights = load_flights(data_file)
169     else:
170         flights = []
171     if args.command == "add":
172         flights = get_flight(
173             flights,
174             args.flight_dest,
175             args.number,
176             args.type
177         )
178     is_dirty = True
179     elif args.command == "display":
180         display_flights(flights)
181     elif args.command == "select":
182         selected = select_flights(flights, args.type)
183         display_flights(selected)
184     if is_dirty:
185         save_flights(data_file, flights)
186
187
188     if __name__ == '__main__':
189         main()
190
```

Рисунок 5 – Код программы, продолжение

```

PS C:\Users\Evil\PycharmProjects\LR #6\ind_task> python idz1.py add --data="flights.json" --flight_dest="Argtjds" --number="AR362" --type="Passenger"
PS C:\Users\Evil\PycharmProjects\LR #6\ind_task> python idz1.py add --flight_dest="Kazakhstan" --number="KZ189" --type="Passenger"
PS C:\Users\Evil\PycharmProjects\LR #6\ind_task> python idz1.py display
usage: flights [-h] [--version] {add,display,select} ...
flights: error: argument command: invalid choice: 'display' (choose from 'add', 'display', 'select')
PS C:\Users\Evil\PycharmProjects\LR #6\ind_task> python idz1.py display
+-----+-----+-----+-----+
| No | Пункт назначения | Номер рейса | Тип самолета |
+-----+-----+-----+-----+
| 1 | Argtjds | AR362 | Passenger |
| 2 | Kazakhstan | KZ189 | Passenger |
+-----+-----+-----+-----+
PS C:\Users\Evil\PycharmProjects\LR #6\ind_task> python idz1.py add --flight_dest="Hdfjdjd" --number="HM234" --type="Military"
PS C:\Users\Evil\PycharmProjects\LR #6\ind_task> python idz1.py display
+-----+-----+-----+-----+
| No | Пункт назначения | Номер рейса | Тип самолета |
+-----+-----+-----+-----+
| 1 | Argtjds | AR362 | Passenger |
| 2 | Kazakhstan | KZ189 | Passenger |
| 3 | Hdfjdjd | HM234 | Military |
+-----+-----+-----+-----+
PS C:\Users\Evil\PycharmProjects\LR #6\ind_task> python idz1.py select military
usage: flights select [-h] [-d DATA] -t TYPE
flights select: error: the following arguments are required: -t/--type
PS C:\Users\Evil\PycharmProjects\LR #6\ind_task> python idz1.py select --type="Military"
+-----+-----+-----+-----+
| No | Пункт назначения | Номер рейса | Тип самолета |
+-----+-----+-----+-----+
| 1 | Hdfjdjd | HM234 | Military |
+-----+-----+-----+-----+
PS C:\Users\Evil\PycharmProjects\LR #6\ind_task> 

```

Рисунок 6 – Результат работы программы

Индивидуальное задание №2

```
idz2.py x db_backup.json x .env x
1  ▶  #!/usr/bin/env python3
2      # -*- coding: utf-8 -*-
3
4      import argparse
5          import json
6          import os.path
7          import sys
8      from dotenv import load_dotenv
9
10
11  def get_flight(fls, dest, num, type):
12      """
13      Добавить данные о работнике.
14      """
15      fls.append(
16          {
17              "flight_destination": dest,
18              "flight_number": num,
19              "airplane_type": type
20          }
21      )
22      return fls
23
24
25  def display_flights(flights):
26      """
27      Отобразить список рейсов
28      """
29      if flights:
30          line = '+-{}-+-{}-+-{}-+-{}-+'.format(
31              '-' * 4,
32              '-' * 30,
33              '-' * 20,
34              '-' * 15
35          )
36          print(line)
37          print(
38              '| {:^4} | {:^30} | {:^20} | {:^15} |'.format(
39                  "No",
```

Рисунок 7 – Код программы

```

40         "Пункт назначения",
41         "Номер рейса",
42         "Тип самолета"
43     )
44 )
45 print(line)
46 for idx, flight in enumerate(flights, 1):
47     print(
48         '| {:>4} | {:<30} | {:<20} | {:<15} |'.format(
49             idx,
50             flight.get('flight_destination', ''),
51             flight.get('flight_number', ''),
52             flight.get('airplane_type', 0)
53         )
54     )
55     print(line)
56
57 else:
58     print("Список рейсов пуст")
59
60
61 def select_flights(flights, airplane_type):
62     """
63     Выбрать рейсы самолётов заданного типа
64     """
65     count = 0
66     res = []
67     for flight in flights:
68         if flight.get('airplane_type') == airplane_type:
69             count += 1
70             res.append(flight)
71     if count == 0:
72         print("рейсы не найдены")
73
74     return res
75
76
77 def save_flights(file_name, fls):
78     """

```

Рисунок 8 – Код программы, продолжение

```
idz2.py × db_backup.json × .env ×
79     Сохранить все записи полётов в файл JSON.
80     """
81     # Открыть файл с заданным именем для записи.
82     with open(file_name, "w", encoding="utf-8") as fout:
83         # Выполнить сериализацию данных в формат JSON.
84         # Для поддержки кириллицы установим ensure_ascii=False
85         json.dump(fls, fout, ensure_ascii=False, indent=4)
86
87
88     def load_flights(file_name):
89         """
90         Загрузить все записи полётов из файла JSON.
91         """
92         # Открыть файл с заданным именем для чтения.
93         with open(file_name, "r", encoding="utf-8") as fin:
94             return json.load(fin)
95
96
97     def main(command_line=None):
98         """
99         Главная функция программы
100        """
101        # Создать родительский парсер для определения имени файла.
102        file_parser = argparse.ArgumentParser(add_help=False)
103        file_parser.add_argument(
104            "-d",
105            "--data",
106            action="store",
107            required=False,
108            help="The data file name"
109        )
110        parser = argparse.ArgumentParser("flights")
111        parser.add_argument(
112            "--version",
113            action="version",
114            version=f"%(prog)s 0.1.0"
115        )
116        subparsers = parser.add_subparsers(dest="command")
117        add = subparsers.add_parser(
```

Рисунок 9 – Код программы, продолжение

```
idz2.py × db_backup.json × .env ×
118     "add",
119     parents=[file_parser],
120     help="Add a new flight"
121 )
122 add.add_argument(
123     "-fld",
124     "--flight_dest",
125     action="store",
126     required=True,
127     help="The flight destination"
128 )
129 add.add_argument(
130     "-n",
131     "--number",
132     action="store",
133     help="The flight number"
134 )
135 add.add_argument(
136     "-t",
137     "--type",
138     action="store",
139     required=True,
140     help="The airplane type"
141 )
142 _ = subparsers.add_parser(
143     "display",
144     parents=[file_parser],
145     help="Display all flights"
146 )
147 select = subparsers.add_parser(
148     "select",
149     parents=[file_parser],
150     help="Select the flights"
151 )
152 select.add_argument(
153     "-t",
154     "--type",
155     action="store",
156     required=True,
```

Рисунок 10 – Код программы, продолжение

```
idz2.py × db_backup.json × .env ×
157         help="The required flight type"
158     )
159     dotenv_path = os.path.join(os.path.dirname(__file__), '.env')
160     if os.path.exists(dotenv_path):
161         load_dotenv(dotenv_path)
162     args = parser.parse_args(command_line)
163     data_file = args.data
164     if not data_file:
165         data_file = os.environ.get("WORKERS_DATA")
166     if not data_file:
167         print("The data file name is absent", file=sys.stderr)
168         sys.exit(1)
169     is_dirty = False
170     if os.path.exists(data_file):
171         flights = load_flights(data_file)
172     else:
173         flights = []
174     if args.command == "add":
175         flights = get_flight(
176             flights,
177             args.flight_dest,
178             args.number,
179             args.type
180         )
181         is_dirty = True
182     elif args.command == "display":
183         display_flights(flights)
184     elif args.command == "select":
185         selected = select_flights(flights, args.type)
186         display_flights(selected)
187     if is_dirty:
188         save_flights(data_file, flights)
189
190
191 if __name__ == '__main__':
192     main()
193
```

Рисунок 11 – Код программы, продолжение

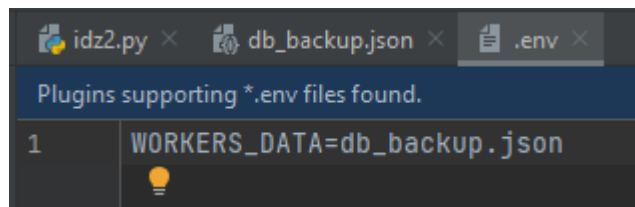
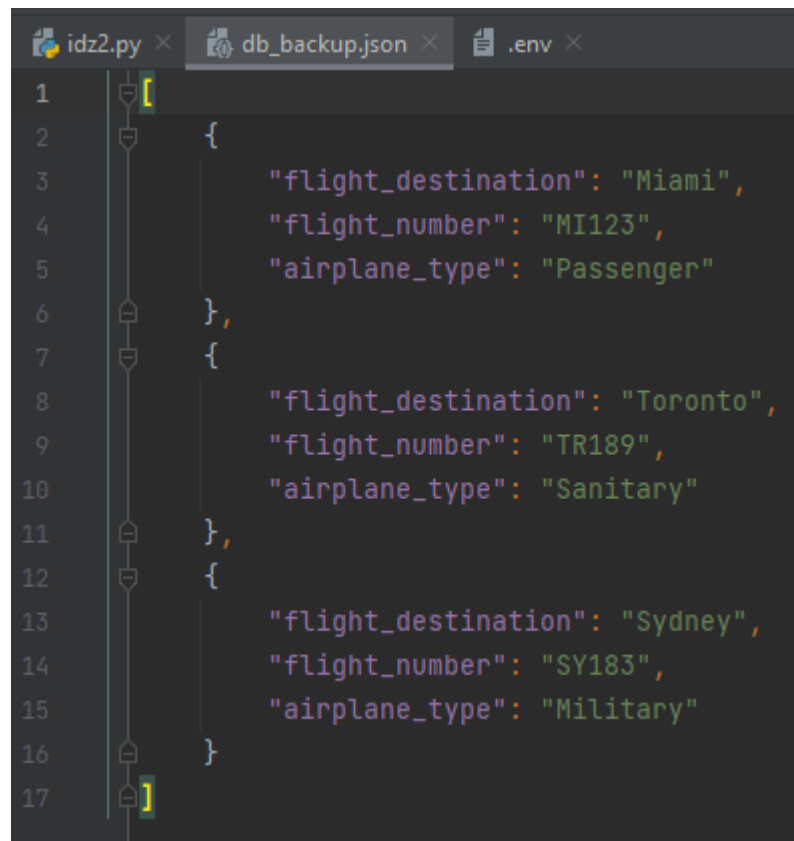


Рисунок 12 – Содержимое файла .env

```
PS C:\Users\Evil\PycharmProjects\LR #6\ind_task2> python idz2.py add --flight_dest="Miami" --number="MI123" --type="Passenger"
PS C:\Users\Evil\PycharmProjects\LR #6\ind_task2> python idz2.py display
+-----+-----+-----+-----+
| No | Пункт назначения | Номер рейса | Тип самолета |
+-----+-----+-----+-----+
| 1 | Miami | MI123 | Passenger |
+-----+-----+-----+-----+
PS C:\Users\Evil\PycharmProjects\LR #6\ind_task2> python idz2.py add --flight_dest="Toronto" --number="TR189" --type="Sanitary"
PS C:\Users\Evil\PycharmProjects\LR #6\ind_task2> python idz2.py add --flight_dest="Sydney" --number="SY183" --type="Military"
PS C:\Users\Evil\PycharmProjects\LR #6\ind_task2> python idz2.py display
+-----+-----+-----+-----+
| No | Пункт назначения | Номер рейса | Тип самолета |
+-----+-----+-----+-----+
| 1 | Miami | MI123 | Passenger |
| 2 | Toronto | TR189 | Sanitary |
| 3 | Sydney | SY183 | Military |
+-----+-----+-----+-----+
PS C:\Users\Evil\PycharmProjects\LR #6\ind_task2> python idz2.py select --type="Sanitary"
+-----+-----+-----+-----+
| No | Пункт назначения | Номер рейса | Тип самолета |
+-----+-----+-----+-----+
| 1 | Toronto | TR189 | Sanitary |
+-----+-----+-----+-----+
PS C:\Users\Evil\PycharmProjects\LR #6\ind_task2>
```

Рисунок 13 – Результат выполнения программы



```
1  [
2  {
3      "flight_destination": "Miami",
4      "flight_number": "MI123",
5      "airplane_type": "Passenger"
6  },
7  {
8      "flight_destination": "Toronto",
9      "flight_number": "TR189",
10     "airplane_type": "Sanitary"
11  },
12  {
13     "flight_destination": "Sydney",
14     "flight_number": "SY183",
15     "airplane_type": "Military"
16  }
17 ]
```

Рисунок 14 – Содержимое файла для сохранения

3. Ответы на вопросы

1. Каково назначение переменных окружения?

Переменная среды (переменная окружения) – это короткая ссылка на какой-либо объект в системе. С помощью таких сокращений, например, можно создавать универсальные пути для приложений, которые будут работать на любых ПК, независимо от имен пользователей и других параметров.

2. Какая информация может храниться в переменных окружения?

Она хранят в текстовом виде ссылки на определённые каталоги, количество ядер процессора, доступные расширения и т.п.

3. Как получить доступ к переменным окружения в ОС Windows?

Просмотреть все переменные окружения можно с помощью команды:
set > %homepath%\desktop\set.txt

Также доступ к ним можно получить через свойства системы.

4. Каково назначение переменных PATH и PATHEXT?

«PATH» позволяет запускать исполняемые файлы и скрипты, «лежащие» в определенных каталогах, без указания их точного местоположения. Например, если ввести в «Командную строку» explorer.exe система осуществит поиск по папкам, указанным в значении переменной, найдет и запустит соответствующую программу.

PATHEXT, в свою очередь, дает возможность не указывать даже расширение файла, если оно прописано в ее значениях.

5. Как создать или изменить переменную окружения в Windows?

- Нажимаем кнопку Создать. Сделать это можно как в пользовательском разделе, так и в системном.
- Вводим имя, например, desktop. Обратите внимание на то, чтобы такое название еще не было использовано (просмотрите списки).
- В поле Значение указываем путь до папки Рабочий стол:
- C:\Users\Имя_пользователя\Desktop
- Нажимаем ОК. Повторяем это действие во всех открытых окнах (см. выше).
- Перезапускаем Проводник и консоль или целиком систему.
- Готово, новая переменная создана, увидеть ее можно в соответствующем списке.

6. Что представляют собой переменные окружения в ОС Linux?

Переменные окружения в Linux представляют собой набор именованных значений, используемых другими приложениями. Переменные окружения применяются для настройки поведения приложений и работы самой системы. Например, переменная окружения может хранить информацию о путях к исполняемым файлам, заданном по умолчанию текстовом редакторе, браузере, языковых параметрах (локали) системы или настройках раскладки клавиатуры.

7. В чем отличие переменных окружения от переменных оболочки?

Переменные окружения (или «переменные среды») — это переменные, доступные в масштабах всей системы и наследуемые всеми дочерними процессами и оболочками.

Переменные оболочки — это переменные, которые применяются только к текущему экземпляру оболочки. Каждая оболочка, например, bash или zsh, имеет свой собственный набор внутренних переменных.

8. Как вывести значение переменной окружения в Linux?

Наиболее часто используемая команда для вывода переменных окружения — `printenv`. Если команде в качестве аргумента передать имя переменной, то будет отображено значение только этой переменной. Если же вызвать `printenv` без аргументов, то выведется построчный список всех переменных окружения.

Пример: `$ printenv HOME`

9. Какие переменные окружения Linux Вам известны?

- `USER` — текущий пользователь.
- `PWD` — текущая директория.
- `OLDPWD` — предыдущая рабочая директория. Используется оболочкой для того, чтобы вернуться в предыдущий каталог при выполнении команды `cd -`.
- `HOME` — домашняя директория текущего пользователя.
- `SHELL` — путь к оболочке текущего пользователя (например, `bash` или `zsh`).
- `EDITOR` — заданный по умолчанию редактор. Этот редактор будет вызываться в ответ на команду `edit`.
- `LOGNAME` — имя пользователя, используемое для входа в систему.
- `PATH` — пути к каталогам, в которых будет производиться поиск вызываемых команд. При выполнении команды система будет проходить по данным каталогам в указанном порядке и выберет первый из них, в котором будет находиться исполняемый файл искомой команды.
- `LANG` — текущие настройки языка и кодировки.
- `TERM` — тип текущего эмулятора терминала.
- `MAIL` — место хранения почты текущего пользователя.
- `LS_COLORS` — задает цвета, используемые для выделения объектов (например, различные типы файлов в выводе команды `ls` будут выделены разными цветами).

10. Какие переменные оболочки Linux Вам известны?

- `BASHOPTS` — список задействованных параметров оболочки, разделенных двоеточием.
- `BASH_VERSION` — версия запущенной оболочки `bash`.
- `COLUMNS` — количество столбцов, которые используются для отображения выходных данных.
- `DIRSTACK` — стек директорий, к которому можно применять команды `pushd` и `popd`.

- HISTFILESIZE — максимальное количество строк для файла истории команд.
- HISTSIZE — количество строк из файла истории команд, которые можно хранить в памяти.
- HOSTNAME — имя текущего хоста.
- IFS — внутренний разделитель поля в командной строке (по умолчанию используется пробел).
- PS1 — определяет внешний вид строки приглашения ввода новых команд.
- PS2 — вторичная строка приглашения.
- SHELL_OPTS — параметры оболочки, которые можно устанавливать с помощью команды set .
- UID — идентификатор текущего пользователя.

11. Как установить переменные оболочки в Linux?

Чтобы создать новую переменную оболочки с именем, например, NEW_VAR и значением Ravesli.com , просто введите:

```
$ NEW_VAR='Ravesli.com'
```

Вы можете убедиться, что переменная действительно была создана, с помощью команды echo :

```
$ echo $NEW_VAR
```

12. Как установить переменные окружения в Linux?

Команда export используется для задания переменных окружения. С помощью данной команды мы экспортируем указанную переменную, в результате чего она будет видна во всех вновь запускаемых дочерних командных оболочках. Переменные такого типа принято называть внешними.

Для создания переменной окружения экспортируем нашу недавно созданную переменную оболочки:

```
$ export NEW_VAR
```

Вы также можете использовать и следующую конструкцию для создания переменной окружения:

```
$ export MY_NEW_VAR="My New Var"
```

Примечание: Созданные подобным образом переменные окружения доступны только в текущем сеансе. Если вы откроете новую оболочку или выйдете из системы, то все переменные будут потеряны.

Если вы хотите, чтобы переменная сохранялась после закрытия сеанса оболочки, то необходимо прописать её в специальном файле. Прописать переменную можно как для текущего пользователя, так и для всех пользователей.

Чтобы установить постоянную переменную окружения для текущего пользователя, откройте файл `.*bashrc`:

```
$ sudo nano ~/.bashrc
```

Для каждой переменной, которую вы хотите сделать постоянной, добавьте в конец файла строку, используя следующий синтаксис:

```
export [ИМЯ_ПЕРЕМЕННОЙ]=[ЗНАЧЕНИЕ_ПЕРЕМЕННОЙ]
```

13. Для чего необходимо делать переменные окружения Linux постоянными?

Для того, чтобы они сохранялись при перезапуске сессии.

14. Для чего используется переменная окружения PYTHONHOME ?

Переменная среды PYTHONHOME изменяет расположение стандартных библиотек Python. По умолчанию библиотеки ищутся в `prefix/lib/pythonversion` и `exec_prefix/lib/pythonversion`, где `prefix` и `exec_prefix` - это каталоги, зависящие от установки, оба каталога по умолчанию - `/usr/local`. Когда для PYTHONHOME задан один каталог, его значение заменяет `prefix` и `exec_prefix`.

Чтобы указать для них разные значения, установите для PYTHONHOME значение `prefix:exec_prefix`.

15. Для чего используется переменная окружения PYTHONPATH ?

Переменная среды PYTHONPATH изменяет путь поиска по умолчанию для файлов модуля. Формат такой же, как для оболочки PATH: один или несколько путей к каталогам, разделенных `os.pathsep` (например, двоеточие в Unix или точка с запятой в Windows). Несуществующие каталоги игнорируются.

Помимо обычных каталогов, отдельные записи PYTHONPATH могут относиться к zip-файлам, содержащим чистые модули Python в исходной или скомпилированной форме. Модули расширения нельзя импортировать из zip-файлов. Путь поиска по умолчанию зависит от установки Python, но обычно начинается с префикса `/lib/pythonversion`. Он всегда добавляется к PYTHONPATH

16. Какие еще переменные окружения используются для управления работой интерпретатора Python?

PYTHONSTARTUP :

Если переменная среды PYTHONSTARTUP это имя файла, то команды Python в этом файле выполняются до отображения первого приглашения в интерактивном режиме. Файл выполняется в том же пространстве имен, в котором выполняются интерактивные команды, так что определенные или импортированные в нем объекты можно использовать без квалификации в интерактивном сеансе.

При запуске вызывает событие аудита `cpython.run_startup` с именем файла в качестве аргумента.

PYTHONOPTIMIZE :

Если в переменной среды PYTHONOPTIMIZE задана непустая строка, это эквивалентно указанию параметра `-O` . Если установлено целое число, то это эквивалентно указанию `-OO` .

PYTHONBREAKPOINT :

Если переменная среды PYTHONBREAKPOINT установлена, то она определяет вызываемый объект с помощью точечной нотации. Модуль, содержащий вызываемый объект, будет импортирован, а затем вызываемый объект будет запущен реализацией по умолчанию `sys.breakpointhook()` , которая сама вызывается встроенной функцией `breakpoint()` . Если

PYTHONBREAKPOINT не задан или установлен в пустую строку, то это эквивалентно значению `pdb.set_trace` . Установка этого значения в строку `0` приводит к тому, что стандартная реализация `sys.breakpointhook()` ничего не делает, кроме немедленного возврата.

PYTHONDEBUG :

Если значение переменной среды PYTHONDEBUG непустая строка, то это эквивалентно указанию опции `-d` . Если установлено целое число, то это эквивалентно многократному указанию `-dd` .

PYTHONINSPECT :

Если значение переменной среды PYTHONINSPECT непустая строка, то это эквивалентно указанию параметра `-i` . Эта переменная также может быть изменена кодом Python с помощью `os.environ` для принудительного режима проверки при завершении программы.

PYTHONUNBUFFERED :

Если значение переменной среды PYTHONUNBUFFERED непустая строка, то это эквивалентно указанию параметра `-u` .

PYTHONVERBOSE :

Если значение переменной среды PYTHONVERBOSE непустая строка, то это эквивалентно указанию опции -v . Если установлено целое число, это эквивалентно многократному указанию - v .

PYTHONCASEOK :

Если значение переменной среды PYTHONCASEOK установлено, то Python игнорирует регистр символов в операторах импорта. Это работает только в Windows и OS X.

PYTHONDONTWRITEBYTECODE :

Если значение переменной среды PYTHONDONTWRITEBYTECODE непустая строка, то Python не будет пытаться писать файлы .рус при импорте исходных модулей. Это эквивалентно указанию параметра -B .

PYTHONPYCACHEPREFIX :

Если значение переменной среды PYTHONPYCACHEPREFIX установлено, то Python будет записывать файлы .рус в зеркальном дереве каталогов по этому пути, а не в каталогах __pycache__ в исходном дереве. Это эквивалентно указанию параметра -X pycache_prefix=PATH .

PYTHONHASHSEED :

Если значение переменной среды PYTHONHASHSEED не установлено или имеет значение random , то случайное значение используется для заполнения хэшей объектов str и bytes .

Если для PYTHONHASHSEED задано целочисленное значение, то оно используется как фиксированное начальное число для генерации hash() типов, охватываемых рандомизацией хэша.

Цель - разрешить повторяемое хеширование, например, для самотестирования самого интерпретатора, или позволить кластеру процессов Python совместно использовать хеш- значения.

Целое число должно быть десятичным числом в диапазоне [0,4294967295]. Указание значения 0 отключит рандомизацию хэша.

PYTHONIOENCODING :

Если значение переменной среды PYTHONIOENCODING установлено до запуска интерпретатора, то оно переопределяет кодировку, используемую для stdin / stdout / stderr , в синтаксисе encodingname:errorhandler . И имя кодировки encodingname , и части :errorhandler являются необязательными и имеют то же значение, что и в функции str.encode() .

Для stderr часть :errorhandler игнорируется, а обработчик всегда будет заменять обратную косую черту.

PYTHONNOUSERSITE :

Если значение переменной среды PYTHONNOUSERSITE установлено, то Python не будет добавлять пользовательский каталог site-packages в переменную sys.path .

PYTHONUSERBASE :

Переменная среды PYTHONUSERBASE определяет базовый каталог пользователя, который используется для вычисления пути к каталогу пользовательских пакетов сайта site-packages и путей установки Distutils для python setup.py install --user .

PYTHONWARNINGS :

Переменная среды PYTHONWARNINGS эквивалентна опции -W . Если она установлена в виде строки, разделенной запятыми, то это эквивалентно многократному указанию -W , при этом фильтры, расположенные позже в списке, имеют приоритет над фильтрами ранее в списке.

В простейших настройках определенное действие безоговорочно применяется ко всем предупреждениям, выдаваемым процессом (даже к тем, которые по умолчанию игнорируются):

PYTHONWARNINGS=default - предупреждает один раз для каждого вызова;

PYTHONWARNINGS=error - преобразовывает в исключения;

PYTHONWARNINGS=always - предупреждает каждый раз;

PYTHONWARNINGS=module - предупреждает один раз для каждого вызванного модуля;

PYTHONWARNINGS=once - предупреждает один раз для каждого процесса Python;

PYTHONWARNINGS=ignore - никогда не предупреждает.

PYTHONFAULTHANDLER :

Если значение переменной среды PYTHONFAULTHANDLER непустая строка, то при запуске вызывается faulthandler.enable() : устанавливается обработчик сигналов SIGSEGV , SIGFPE ,

SIGABRT , SIGBUS и SIGILL , чтобы вывести данные трассировки Python. Это эквивалентно опции обработчика ошибок -X .

PYTHONTRACEMALLOC :

Если значение переменной среды PYTHONTRACEMALLOC непустая строка, то начнется отслеживание выделения памяти Python с помощью модуля tracemalloc . Значение переменной - это максимальное количество кадров, хранящихся в обратной трассировке trace .

Например, PYTHONTRACEMALLOC=1 сохраняет только самый последний кадр.

PYTHONPROFILEIMPORTTIME :

Если значение переменной среды PYTHONPROFILEIMPORTTIME непустая строка, то Python покажет, сколько времени занимает каждый импорт. Это в точности эквивалентно установке -X importtime в командной строке.

PYTHONASYNCIODEBUG :

Если значение переменной среды PYTHONASYNCIODEBUG непустая строка, то включается режим отладки модуля asyncio .

PYTHONMALLOC :

Переменная PYTHONMALLOC задает распределители памяти Python и/или устанавливает отладочные хуки. Задает семейство распределителей памяти, используемых Python:

default : использует распределители памяти по умолчанию.

malloc : использует функцию malloc() библиотеки C для всех доменов (PYMEM_DOMAIN_RAW , PYMEM_DOMAIN_MEM , PYMEM_DOMAIN_OBJ).

rumalloc : использует распределитель rumalloc для доменов PYMEM_DOMAIN_MEM и PYMEM_DOMAIN_OBJ и использует функцию malloc() для домена PYMEM_DOMAIN_RAW .

Устанавливает хуки отладки:

debug : устанавливает хуки отладки поверх распределителей памяти по умолчанию.

malloc_debug : то же, что и malloc , но также устанавливает отладочные хуки.

`rumalloc_debug` : то же, что и `rumalloc` , но также устанавливает отладочные хуки.

PYTHONMALLOCSTATS :

Если значение переменной среды `PYTHONMALLOCSTATS` непустая строка, то Python будет печатать статистику распределителя памяти `rumalloc` каждый раз, когда создается новая область объекта `rumalloc` , а также при завершении работы.

Эта переменная игнорируется, если переменная среды `PYTHONMALLOC` используется для принудительного использования распределителя `malloc()` библиотеки C или если Python настроен без поддержки `rumalloc` .

PYTHONLEGACYWINDOWSFSENCODING :

Если значение переменной среда Python `PYTHONLEGACYWINDOWSFSENCODING` непустая строка, то кодировка файловой системы по умолчанию и режим ошибок вернутся к своим значениям `mbcs` и `replace` до версии Python 3.6 соответственно. В противном случае используются новые значения по умолчанию `utf-8` и `surrogatepass` .

PYTHONLEGACYWINDOWSSTDIO :

Если значение переменной среды `PYTHONLEGACYWINDOWSSTDIO` непустая строка, то новые средства чтения и записи консоли не используются. Это означает, что символы Unicode будут закодированы в соответствии с активной кодовой страницей консоли, а не с использованием `utf-8` .

Эта переменная игнорируется, если стандартные потоки перенаправляются в файлы или каналы, а не ссылаются на буферы консоли.

PYTHONCOERCECLOCALE :

Если значение переменной среды `PYTHONCOERCECLOCALE` установлено в значение 0 , то это заставит основное приложение командной строки Python пропускать приведение устаревших локалей C и POSIX на основе ASCII к более функциональной альтернативе на основе UTF-8 . Если эта переменная не установлена или имеет значение, отличное от 0 , то переменная среды переопределения локали `LC_ALL` также не задана, а текущая локаль, указанная для категории

`LC_STYPE` , является либо локалью C по умолчанию, либо локалью POSIX явно основанной на

ASCII , то Python CLI попытается настроить следующие локали для категории LC_STYPE в

порядке, указанном перед загрузкой среды выполнения интерпретатора:

C.UTF-8 ,

C.utf8 ,

UTF-8 .

Если установка одной из этих категорий локали прошла успешно, то переменная среды

LC_STYPE также будет установлена соответствующим образом в текущей среде процесса и инициализации среды выполнения Python. Это гарантирует, что обновленный параметр будет виден как самому интерпретатору, так и другим компонентам, зависящим от локали, работающим в одном процессе (например, библиотеке GNU readline), и в subprocesses (независимо от того, работают ли эти процессы на интерпретаторе Python или нет), а также в операциях, которые запрашивают среду, а не текущую локаль C (например, собственный

locale.getdefaultlocale() Python).

Настройка одного из этих языковых стандартов явно или с помощью указанного выше неявного принуждения языкового стандарта автоматически включает обработчик ошибок surrogateescape для sys.stdin и sys.stdout (sys.stderr продолжает использовать обратную косую черту, как и в любой другой локали). Это поведение обработки потока можно переопределить, используя PYTHONIOENCODING , как обычно.

Для целей отладки, установка PYTHONCOERCECLOCALE=warn приведет к тому, что Python будет выдавать предупреждающие сообщения на stderr , если активируется принуждение языкового стандарта или если языковой стандарт, который мог бы вызвать приведение, все еще активен при инициализации среды выполнения Python. Также обратите внимание, что даже когда принуждение языкового стандарта отключено или когда не удастся найти подходящую целевую локаль, переменная среды PYTHONUTF8 все равно будет активироваться по умолчанию в устаревших локалях на основе ASCII . Чтобы для системных интерфейсов интерпретатор использовал ASCII вместо UTF-8 , необходимо обе переменные отключить.

PYTHONDEVMODE :

Если значение переменной среды PYTHONDEVMODE непустая строка, то включится режим разработки Python, введя дополнительные проверки времени выполнения, которые слишком "дороги" для включения по умолчанию.

PYTHONUTF8 :

Если переменная среды PYTHONUTF8 установлена в значение 1, то это включает режим интерпретатора UTF-8 , где UTF-8 используется как кодировка текста для системных интерфейсов, независимо от текущей настройки локали.

PYTHONWARNDEFAULTENCODING :

Если для этой переменной среды задана непустая строка, то код будет выдавать EncodingWarning , когда используется кодировка по умолчанию, зависящая от локали.

PYTHONTHREADDEBUG :

Если значение переменной среды PYTHONTHREADDEBUG установлено, то Python распечатает отладочную информацию о потоках. Нужен Python, настроенный с параметром сборки --with-pydebug .

PYTHONDUMPPREFS :

Если значение переменной среды PYTHONDUMPPREFS установлено, то Python будет сбрасывать объекты и счетчики ссылок, все еще живые после завершения работы интерпретатора.

17. Как осуществляется чтение переменных окружения в программах на языке программирования Python?

Для начала потребуется импортировать модуль os, чтобы считывать переменные. Для доступа к переменным среды в Python используется объект os.environ . С его помощью программист может получить и изменить значения всех переменных среды. Далее мы рассмотрим различные способы чтения, проверки и присвоения значения переменной среды.

```
# Импортируем модуль os
```

```
import os
```

```
# Создаём цикл, чтобы вывести все переменные среды
```

```
print("The keys and values of all environment variables:")
```

```
for key in os.environ:  
    print(key, '=>', os.environ[key])  
  
# Выводим значение одной переменной  
print("The value of HOME is: ", os.environ['HOME'])
```

18. Как проверить, установлено или нет значение переменной окружения в программах на языке программирования Python?

```
key_value = input("Enter the key of the environment variable:")  
if os.environ[key_value]:
```

19. Как присвоить значение переменной окружения в программах на языке программирования Python?

Для присвоения значения любой переменной среды используется функция `setdefault()`. Давайте напишем код, чтобы с помощью функции `setdefault()` изменить значение переменной `DEBUG` на `True` (по умолчанию установлено `False`). После установки значения мы проверим его функцией `get()`.

Если мы сделали всё правильно, выведется сообщение «Режим отладки включен», в противном случае – «Режим отладки выключен».

```
# Импортируем модуль os  
import os  
  
# Задаём значение переменной DEBUG  
os.environ.setdefault('DEBUG', 'True')  
  
# Проверяем значение переменной  
if os.environ.get('DEBUG') == 'True':  
    print('Debug mode is on')  
else:  
    print('Debug mode is off')
```

Значения переменных окружения можно считывать и изменять при помощи объекта `environ[]` модуля `os` либо путем использования функций `setdefault()` и `get()`. В качестве ключа, по которому можно обратиться и получить либо присвоить значение переменной, в `environ[]` используется имя

переменной окружения. Функция `get()` используется для получения значения определённой переменной, а `setdefault()` – для инициализации.