

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт цифрового развития
Кафедра инфокоммуникаций

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №9
дисциплины «Программирование на Python»
Вариант 29

Выполнил:
Саенко Андрей Максимович
2 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и
вычислительная техника»,
направленность (профиль)
«Программное обеспечение средств
вычислительной техники и
автоматизированных систем», очная
форма обучения

(подпись)

Руководитель практики:
Воронкин Р.А., канд. технических
наук, доцент кафедры
инфокоммуникаций

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____
Ставрополь, 2023 г.

Тема: Работа со словарями в языке Python

Цель: приобретение навыков по работе со словарями при написании программ с помощью языка программирования Python версии 3.x.

Порядок выполнения работы:

1. Проработаны примеры из лабораторной работы.

Пример 1. Использовать словарь, содержащий следующие ключи: фамилия и инициалы работника; название занимаемой должности; год поступления на работу.

Написать программу, выполняющую следующие действия: ввод с клавиатуры данных в список, состоящий из заданных словарей; записи должны быть размещены по алфавиту; вывод на дисплей фамилий работников, чей стаж работы в организации превышает значение, введенное с клавиатуры; если таких работников нет, вывести на дисплей соответствующее сообщение.

Код программы:

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

import sys
from datetime import date

if __name__ == '__main__':
    # Список работников.
    workers = []

    # Организовать бесконечный цикл запроса команд.
    while True:
        # Запросить команду из терминала.
        command = input(">>> ").lower()

        # Выполнить действие в соответствие с командой.
        if command == 'exit':
            break
        elif command == 'add':
            # Запросить данные о работнике.
            name = input("Фамилия и инициалы? ")
            post = input("Должность? ")
            year = int(input("Год поступления? "))

            # Создать словарь.
```

```

worker = {
    'name': name,
    'post': post,
    'year': year,
}

# Добавить словарь в список.
workers.append(worker)
# Отсортировать список в случае необходимости.
if len(workers) > 1:
    workers.sort(key=lambda item: item.get('name', ''))

elif command == 'list':
    # Заголовок таблицы.
    line = '+-{}-+-{}-+-{}-+-{}-+'.format(
        '-' * 4,
        '-' * 30,
        '-' * 20,
        '-' * 8
    )
    print(line)
    print(
        '| {:^4} | {:^30} | {:^20} | {:^8} |'.format(
            "№",
            "Ф.И.О.",
            "Должность",
            "Год"
        )
    )
    print(line)

    # Вывести данные о всех сотрудниках.
    for idx, worker in enumerate(workers, 1):
        print(
            '| {:>4} | {:<30} | {:<20} | {:>8} |'.format(
                idx,
                worker.get('name', ''),
                worker.get('post', ''),
                worker.get('year', 0)
            )
        )
        print(line)

elif command.startswith('select '):
    # Получить текущую дату.
    today = date.today()

    # Разбить команду на части для выделения номера года.
    parts = command.split(' ', maxsplit=1)
    # Получить требуемый стаж.
    period = int(parts[1])

```

```

# Инициализировать счетчик.
count = 0
# Проверить сведения работников из списка.
for worker in workers:
    if today.year - worker.get('year', today.year) >= period:
        count += 1
        print(
            '{:>4}: {}'.format(count, worker.get('name', ''))
        )

# Если счетчик равен 0, то работники не найдены.
if count == 0:
    print("Работники с заданным стажем не найдены.")
elif command == 'help':
    # Вывести справку о работе с программой.
    print("Список команд:\n")
    print("add - добавить работника;")
    print("list - вывести список работников;")
    print("select <стаж> - запросить работников со стажем;")
    print("help - отобразить справку;")
    print("exit - завершить работу с программой.")
else:
    print(f"Неизвестная команда {command}", file=sys.stderr)

```

Результат работы программы:

```

>>> help
Список команд:

add - добавить работника;
list - вывести список работников;
select <стаж> - запросить работников со стажем;
help - отобразить справку;
exit - завершить работу с программой.
>>> add
Фамилия и инициалы? Кузнецов К.К.
Должность? Программист
Год поступления? 1987
>>> add
Фамилия и инициалы? Иванов И.И.
Должность? Программист
Год поступления? 2022
>>> select 10
1: Кузнецов К.К.
>>> list
+-----+-----+-----+-----+
| № | Ф.И.О. | Должность | Год |
+-----+-----+-----+-----+
| 1 | Иванов И.И. | Программист | 2022 |
+-----+-----+-----+-----+
| 2 | Кузнецов К.К. | Программист | 1987 |
+-----+-----+-----+-----+
>>> exit
PS C:\Users\HAIER>

```

Рисунок 1 – Результат работы программы

2. Выполнено индивидуальное задание:

Использовать словарь, содержащий следующие ключи: название начального пункта маршрута; название конечного пункта маршрута; номер маршрута.

Написать программу, выполняющую следующие действия: ввод с клавиатуры данных в список, состоящий из словарей заданной структуры; записи должны быть упорядочены по номерам маршрутов; вывод на экран информации о маршрутах, которые начинаются или оканчиваются в пункте, название которого введено с клавиатуры; если таких маршрутов нет, выдать на дисплей соответствующее сообщение.

```
>>> add
Начальный пункт маршрута? Москва
Конечный пункт маршрута? Новосибирск
Номер маршрута? 1
>>> add
Начальный пункт маршрута? Омск
Конечный пункт маршрута? Москва
Номер маршрута? 2
>>> add
Начальный пункт маршрута? Красноярск
Конечный пункт маршрута? Краснодар
Номер маршрута? 3
>>> select Москва
1: Москва-Новосибирск, номер маршрута: 1
2: Омск-Москва, номер маршрута: 2
>>> list
+-----+-----+-----+
| № | Начальный пункт | Конечный пункт | Номер |
+-----+-----+-----+
| 1 | Москва | Новосибирск | 1 |
+-----+-----+-----+
| 2 | Омск | Москва | 2 |
+-----+-----+-----+
| 3 | Красноярск | Краснодар | 3 |
+-----+-----+-----+
>>> 
```

Рисунок 2 – Результат работы программы

```
>>> select Ставрополь
Маршрут не найден.
>>> help
Список команд:

add - добавить маршрут;
list - вывести список маршрутов;
select <пункт> - запросить информацию о маршруте с указанным пунктом;
help - отобразить справку;
exit - завершить работу с программой.
>>> 
```

Рисунок 3 – Результат работы программы

Код программы:

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

import sys

if __name__ == '__main__':
    routes = []

    while True:

        command = input(">>> ").lower()

        if command == 'exit':
            break
        elif command == 'add':

            name_start = input("Начальный пункт маршрута? ")
            name_end = input("Конечный пункт маршрута? ")
            number = int(input("Номер маршрута? "))

            route = {
                'name_start': name_start,
                'name_end': name_end,
                'number': number,
            }

            routes.append(route)
            if len(routes) > 1:
                routes.sort(key=lambda item: item.get('number', ""))

        elif command == 'list':
            line = '+-{}-+-{}-+-{}-+-{}-+'.format(
                '-' * 4,
                '-' * 30,
                '-' * 30,
                '-' * 8
            )
            print(line)
            print(
                '| {:^4} | {:^30} | {:^30} | {:^8} |'.format(
                    "№",
                    "Начальный пункт",
                    "Конечный пункт",
                    "Номер"
                )
            )
            print(line)

            for idx, route in enumerate(routes, 1):
                print(
```

```

'| {:>4} | {:<30} | {:<30} | {:>8} |'.format(
idx,
route.get('name_start', ''),
route.get('name_end', ''),
route.get('number', 0)
)
)
print(line)

```

```

elif command.startswith('select '):

```

```

    parts = command.split(' ', maxsplit=1)
    station = parts[1]
    count = 0

```

```

    for route in routes:

```

```

        if (station == route["name_start"].lower() or
            station == route["name_end"].lower()):

```

```

            count += 1
            print(
                '{:>4}: {}-{}'.format(count,
                    route["name_start"], route["name_end"], route["number"])
            )

```

```

        if count == 0:
            print("Маршрут не найден.")

```

```

elif command == 'help':

```

```

    print("Список команд:\n")
    print("add - добавить маршрут;")
    print("list - вывести список маршрутов;")
    print("select <пункт> - запросить информацию"+
        " о маршруте с указанным пунктом;")
    print("help - отобразить справку;")
    print("exit - завершить работу с программой.")

```

```

else:

```

```

    print(f"Неизвестная команда {command}", file=sys.stderr)

```

Составлена UML-диаграмма:

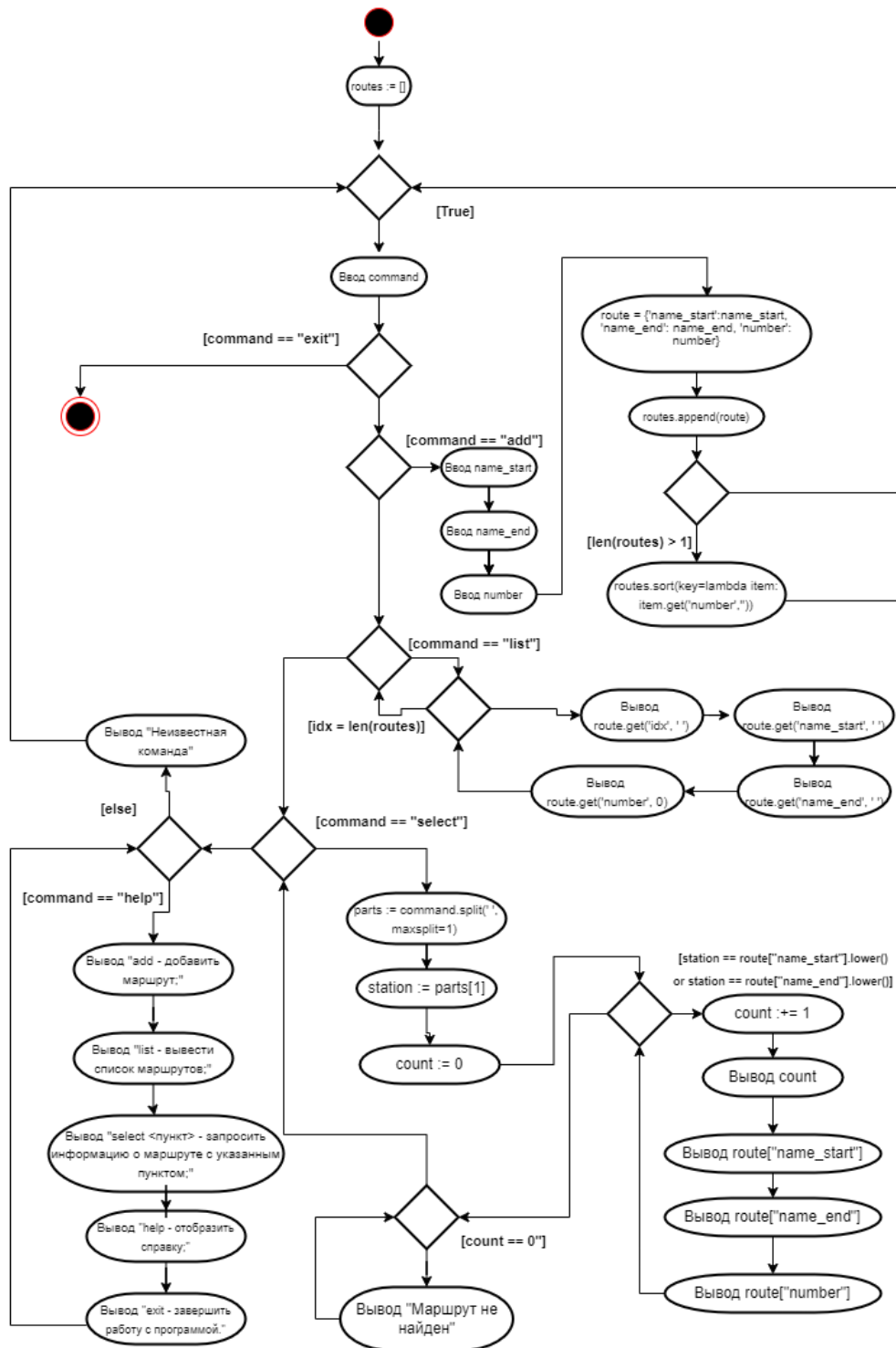


Рисунок 4 – UML-диаграмма

3. Выполнено задание:

Решите задачу: создайте словарь, связав его с переменной school , и наполните данными, которые бы отражали количество учащихся в разных классах (1а, 1б, 2б, 6а, 7в и т. п.). Внесите изменения в словарь согласно следующему: а) в одном из классов изменилось количество учащихся, б) в

школе появился новый класс, с) в школе был расформирован (удален) другой класс. Вычислите общее количество учащихся в школе.

Код программы:

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

if __name__ == '__main__':
    school = {"1а": 20, "1б": 30, "1в": 40}
    print(school)
    #Изменение количества учащихся в одном из классов
    school["1а"] = 10
    print(school)

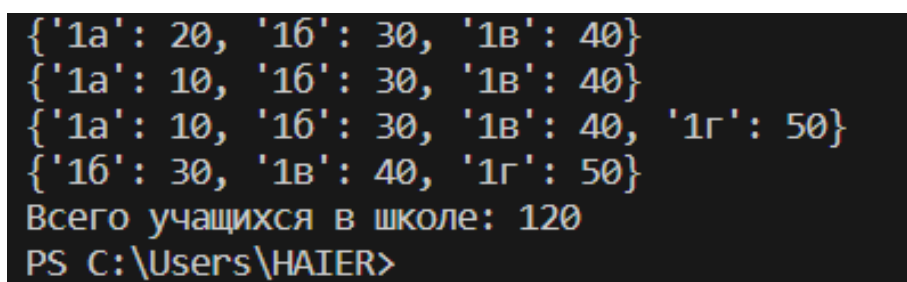
    #Создание нового класса
    school["1г"] = 50
    print(school)

    #Расформирование класса
    school.pop("1а")
    print(school)

    counter = 0
    for i in school.keys():
        counter += school[i]

    print(f"Всего учащихся в школе: {counter}")
```

Результат работы программы:



```
{ '1а': 20, '1б': 30, '1в': 40 }
{ '1а': 10, '1б': 30, '1в': 40 }
{ '1а': 10, '1б': 30, '1в': 40, '1г': 50 }
{ '1б': 30, '1в': 40, '1г': 50 }
Всего учащихся в школе: 120
PS C:\Users\NAIER>
```

Рисунок 5 – Результат работы программы

4. Выполнено задание:

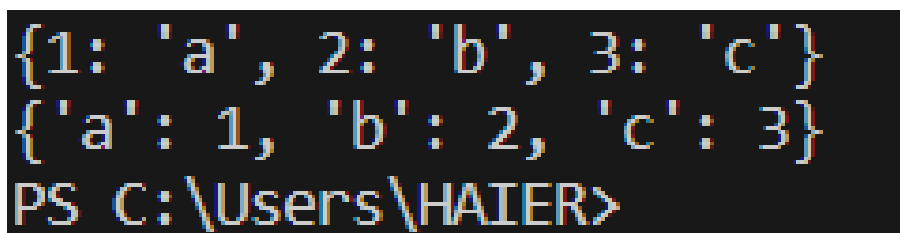
Решите задачу: создайте словарь, где ключами являются числа, а значениями – строки. Примените к нему метод `items()`, с помощью полученного объекта `dict_items` создайте новый словарь, "обратный" исходному, т. е. ключами являются строки, а значениями – числа.

Код программы:

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

if __name__ == '__main__':
    d = {1: "a", 2: "b", 3: "c"}
    print(d)
    swapped = {v: k for k, v in d.items()}
    print(swapped)
```

Результат работы программы:



```
{1: 'a', 2: 'b', 3: 'c'}
{'a': 1, 'b': 2, 'c': 3}
PS C:\Users\HAIER>
```

Рисунок 6 – Результат работы программы

Ответы на контрольные вопросы

1. Что такое словари в языке Python?

Словарь представляет собой структуру данных, предназначенную для хранения произвольных объектов с доступом по ключу.

2. Может ли функция `len()` быть использована при работе со словарями?

Данная функция может быть использована при работе со словарями.

3. Какие методы обхода словарей Вам известны?

Обойти словарь можно с помощью циклов `for` и `while`, либо с помощью словаря включений.

4. Какими способами можно получить значения из словаря по ключу?

В словаре доступ к значениям осуществляется по ключам, которые заключаются в квадратные скобки (по аналогии с индексами списков). Для этого подходит и метод `get()`.

5. Какими способами можно установить значение в словаре по ключу?

Чтобы установить значение в словаре по ключу, нужно обратиться к значению, доступному по этому ключу и использовать оператор присваивания «=».

6. Что такое словарь включений?

Словарь включений аналогичен списковым включениям, за исключением того, что он создаёт объект словаря вместо списка.

7. Самостоятельно изучите возможности функции `zip()` приведите примеры ее использования.

Функция `zip()` в Python создает итератор, который объединяет элементы из нескольких источников данных. Эта функция работает со списками, кортежами, множествами и словарями для создания списков или кортежей, включающих все эти данные.

Пример использования функции:

```
numbers = [1, 2, 3]
chars = ["a", "b", "c"]

zipped_values = zip(numbers, chars)
zipped_list = list(zipped_values)

print(zipped_list)
```

В консоль будет выведена следующая строка:

```
[('a', 1), ('b', 2), ('c', 3)]
```

8. Самостоятельно изучите возможности модуля `datetime`. Каким функционалом по работе с датой и временем обладает этот модуль

Модуль `datetime` предоставляет классы для обработки времени и даты разными способами.

Классы модуля `datetime`:

Класс `datetime.date(year, month, day)` - стандартная дата. Атрибуты: `year`, `month`, `day`. Неизменяемый объект.

Класс `datetime.time(hour=0, minute=0, second=0, microsecond=0, tzinfo=None)` - стандартное время, не зависит от даты. Атрибуты: `hour`, `minute`, `second`, `microsecond`, `tzinfo`.

Класс `datetime.timedelta` - разница между двумя моментами времени, с точностью до микросекунд.

Класс `datetime.tzinfo` - абстрактный базовый класс для информации о временной зоне (например, для учета часового пояса и / или летнего времени).

Класс `datetime.datetime(year, month, day, hour=0, minute=0, second=0, microsecond=0, tzinfo=None)` - комбинация даты и времени.

Методы `datetime`:

`datetime.today()` - объект `datetime` из текущей даты и времени. Работает также, как и `datetime.now()` со значением `tz=None`.

`datetime.fromtimestamp(timestamp)` - дата из стандартного представления времени.

`datetime.fromordinal(ordinal)` - дата из числа, представляющего собой количество дней, прошедших с 01.01.1970.

`datetime.now(tz=None)` - объект `datetime` из текущей даты и времени.

`datetime.combine(date, time)` - объект `datetime` из комбинации объектов `date` и `time`.

`datetime.strptime(date_string, format)` - преобразует строку в `datetime` (так же, как и функция `strptime` из модуля time).

`datetime.strptime(format)` - см. функцию `strptime` из модуля `time`.

`datetime.date()` - объект даты (с отсечением времени).

`datetime.time()` - объект времени (с отсечением даты).

`datetime.replace([year[, month[, day[, hour[, minute[, second[, microsecond[, tzinfo]]]]]]])` - возвращает новый объект `datetime` с изменёнными атрибутами.

`datetime.timetuple()` - возвращает `struct_time` из `datetime`.

`datetime.toordinal()` - количество дней, прошедших с 01.01.1970.

`datetime.timestamp()` - возвращает время в секундах с начала эпохи.

`datetime.weekday()` - день недели в виде числа, понедельник - 0, воскресенье - 6.

`datetime.isoweekday()` - день недели в виде числа, понедельник - 1, воскресенье - 7.

`datetime.isocalendar()` - кортеж (год в формате ISO, ISO номер недели, ISO день недели).

`datetime.isoformat(sep='T')` - красивая строка вида "YYYY-MM-DDTHH:MM:SS.mmmmmm" или, если `microsecond == 0`, "YYYY-MM-DDTHH:MM:SS"

`datetime.ctime()` – преобразует число секунд в дату.

Вывод

В ходе выполнения работы приобретены навыки по работе со словарями при написании программ с помощью языка программирования Python версии 3.x.