13. Работа со словарями

- Словари
- Методы словарей
- Генератор словарей
- Вложенные словари

Одним из наиболее важных составных типов данных в Python является словарь. Это коллекция, в которой данные хранятся в виде пар ключ-значение {key:value, key_2:value}

Словари **упорядочены** и **изменяемы**, а **ключи в них не могут дублироваться**. При этом стоит иметь в виду, что до Python 3.6 версии словари *были не упорядочены*.

- Словари

Словарь состоит из набора пар {ключ:значение}. Двоеточие (:) отделяет каждый ключ от связанного с ним значения

Ключами словаря могут быть только неизменяемые типы данных. Например, строки, числа, кортежи или логические значения(boolean). Кроме того, **ключи должны быть уникальными** и соответственно не могут повторяться

Значения, наоборот, могут быть представлять собой любые типы данных и повторяться.

Создание словаря

Основные операции словарями – **CRUD** (*Create, Read, Update, Delete – Создание, Чтение, Модификация, Удаление*).

Пустой словарь можно инициализировать через фигурные скобки {} или функцию(конструктор) **dict**()

```
empty_dict = {}
empty_dict_2 = dict()

print(empty_dict)
print(empty_dict_2)
```

Рисунок.1 - Создание пустого словаря

```
{}
{}

Process finished with exit code 0
```

Рисунок.1.1 – Результат Рис.1

Словарь можно сразу заполнить ключами и значениями.

```
assistant_yandex = {'name': 'Алиса', 'release_date': '10 октября 2017', 'company': 'Яндекс'}
print(assistant_yandex)
```

Рисунок.1.2 – Создание словаря с исходными значениями

```
{'name': 'Алиса', 'release_date': '10 октября 2017', 'company': 'Яндекс'}

Process finished with exit code 0
```

Рисунок.1.2.1 – Результат Рис.1.2

Словарь можно также создать из вложенных списков (или кортежей)

```
assistant_google = dict([['name', 'Google Assistant'], ['release_date', '18 мая 2016 г'], ['company', 'Google']])
print(assistant_google)
```

Рисунок.1.3 – Создание словаря с исходными значениями из вложенных списков

```
{'name': 'Google Assistant', 'release_date': '18 мая 2016 г', 'company': 'Google'}

Process finished with exit code 0
```

Рисунок.1.3.1 – Результат Рис.1.3

Иногда бывает полезно создать словарь с заранее известными ключами и заданным значением. В этом нам поможет метод **dict.fromkeys()**

```
# ключи словаря
assistant_keys = ('name', 'release_date', 'company')
# значение каждого ключа будет 'Unknown'
assistant_value = 'Unknown'

# если не укажете значения (assistant_value), то все ключи получат значение None
assistant_base = dict.fromkeys(assistant_keys, assistant_value)

print(assistant_base)
```

*Рисунок.1.4 – Создание словаря с использованием метода **dict.fromkeys()** с начальными значениями*

```
{'name': 'Unknown', 'release_date': 'Unknown', 'company': 'Unknown'}

Process finished with exit code 0
```

Рисунок.1.4.1 – Результат Рис.1.4

```
# ключи словаря
assistant_keys = ('name', 'release_date', 'company')
# значение каждого ключа будет 'Unknown'
assistant_value = 'Unknown'

# если не укажете значения (assistant_value), то все ключи получат значение None
assistant_base_2 = dict.fromkeys(assistant_keys)
print(assistant_base_2)
```

*Рисунок.1.5 – Создание словаря с использованием метода **dict.fromkeys()** без указания начального значения*

```
{'name': None, 'release_date': None, 'company': None}

Process finished with exit code 0
```

Рисунок.1.5.1 – Результат Рис.1.5

Доступ к значениям словаря

Конкретное значение в словаре можно получить, введя название словаря и затем название ключа в квадратных скобках

```
print(assistant_yandex['name'])

# если такого ключа нет, Python выдаст ошибку KeyError 'age'
print(assistant_google['age'])
```

Рисунок.2 – Получить значение из словаря по ключу

```
    print(assistant_google['age'])
KeyError: 'age'

Алиса

Process finished with exit code 1
```

Рисунок.2.1 – Результат Рис.2

Но мы также можем избежать этой ошибки, используя метод **dict.get()** Он также выводит значение по ключу.Если ключ существует в словаре, мы получим соответствующее значение. А если такого ключа нет, то код не выдаст нам ошибку, а возвращает значение **None**.

```
print(assistant_yandex.get('name'))
print(assistant_google.get('age'))
```

*Рисунок.2.2 – Получить значение из словаря с использованием метода **dict.get(key)***

```
Алиса
None

Process finished with exit code 0
```

Рисунок.2.2.1 – Результат Рис.2.2

В метод **dict.get()** можем вместе с ключом передать второй аргумент, в случае, когда ключ не найден, вместо **None** возвращается значение по умолчанию (*значение, которое мы передали методу **dict.get(key, v)** в качестве второго аргумента*)

```
print(assistant_yandex.get('name'))
print(assistant_google.get('age', 'Not found'))
```

*Рисунок.2.3 – Получить значение из словаря с использованием метода **dict.get(key, v)***

```
Алиса
Not found

Process finished with exit code 0
```

Рисунок.2.3.1 – Результат Рис.2.3

Проверка наличия ключа и значения в словаре

С помощью оператора **in** мы можем проверить наличие определенного ключа в словаре

```
>>> assistant_yandex = {'name': 'Алиса', 'release_date': '10 октября 2017', 'company': 'Яндекс'}
>>> assistant_google = dict([['name', 'Google Assistant'], ['release_date', '18 мая 2016 г'], ['company', 'Google']])
>>> 'company' in assistant_google
True
>>> 'last_name' in assistant_yandex
False
```

*Рисунок.3 - проверка наличия ключа с помощью оператора **in***

Важно сказать, что оператор **in** работает быстрее метода **dict.get()**

Используя метод **dict.values()** можно проверить наличие определенного значения

```
>>> 'Amazon' in assistant_google.values()
False
>>> 'Яндекс' in assistant_yandex.values()
True
```

*Рисунок.3.1 - проверка наличия определенного значения с помощью оператора **in** и метода **dict.values()***

С помощью метода **dict.items()** можем проверить наличие пары *ключ:значение*

```
>>> ('company', 'Google') in assistant_google.items()
True
>>> ('name', 'Siri') in assistant_yandex.items()
False
```

*Рисунок.3.2 - проверка наличия пары **ключ:значение** с помощью оператора **in** и метода **dict.items()***

Добавление и изменение (обновление) элементов

Добавить элемент можно, передав новому ключу новое значение.

```
assistant_yandex['languages'] = ['Русский']

print(assistant_yandex)
```

Рисунок.4 – Добавление нового ключа и значения в словарь

Обратите внимание, в данном случае новое значение – это список

```
{'name': 'Алиса', 'release_date': '10 октября 2017', 'company': 'Яндекс', 'languages': ['Русский']}

Process finished with exit code 0
```

Рисунок.4.1 – Результат Рис.4

Изменить элемент можно передав существующему ключу новое значение.

```
assistant_yandex['languages'] = ['Английский', 'Русский', 'Китайский', 'Французский']

print(assistant_yandex)
```

Рисунок.4.2 – Изменение значения существующего ключа

```
{'name': 'Алиса', 'release_date': '10 октября 2017', 'company': 'Яндекс', 'languages': ['Английский', 'Русский', 'Китайский', 'Французский']}

Process finished with exit code 0
```

Рисунок.4.2.1 – Результат Рис.4.2

Начиная с версии *Python 3.9*, в языке появились новые операторы, которые облегчают процесс слияния словарей.

- Merge **|** – это оператор позволяет объединять два словаря с помощью одного символа
- Update **|=** – с помощью такого оператора можно обновить первый словарь значением второго (с типом **dict**)

Основные отличия этих двух операторов:

- **|** создает новый словарь, объединяя два, а **|=** обновляет первый словарь
- Оператор merge **|** упрощает процесс объединения словарей и работы с их значениями
- Оператор update **|=** используется для обновления словарей

```python
# Оператор merge |
dict_1 = {'brand': 'DELL', 'founded': 1984}
dict_2 = {'brand': 'Hewlett-Packard', 'founded': 1939}
dict_3 = dict_1 | dict_2

print(dict_3)
```

Рисунок.4.3 – Оператор merge **|**

```
{'brand': 'Hewlett-Packard', 'founded': 1939}

Process finished with exit code 0
```

Рисунок.4.3.1 – Результат Рис.4.3

```python
# Оператор update |=
dict_1 = {'brand': 'DELL', 'founded': 1984}
dict_2 = {'founder': 'Michael Dell', 'city': 'USA'}
dict_2 |= dict_1
print(dict_2)
```

Рисунок.4.4 – Оператор update **|=**

```
{'founder': 'Michael Dell', 'city': 'USA', 'brand': 'DELL', 'founded': 1984}

Process finished with exit code 0
```

Рисунок.4.4.1 – Результат Рис.4.4

Примечание: при наличии пересекающихся ключей (*а в словарях Python может быть только один уникальный ключ*) останется ключ второго словаря, а первый просто заменится.

Удаление элементов

Есть несколько способов удалить элементы из словаря

```
del dict_2['brand']
print(dict_2)
```

*Рисунок.5 – Удаление элемента с помощью ключевого слово **del***

```
{'founder': 'Michael Dell', 'city': 'USA', 'founded': 1984}

Process finished with exit code 0
```

Рисунок.5.1 – Результат Рис.5

```
var = dict_2.pop('founder')
print(dict_2)
print(var)  # pop удаляет элемент по ключу и возвращает его значение
```

*Рисунок.5.2 – Удаление элемента с помощью метода **dict.pop(key)***

```
{'city': 'USA', 'founded': 1984}
Michael Dell

Process finished with exit code 0
```

Рисунок.5.2.1 – Результат Рис.5.2

Ключевое слово **del** также позволяет удалить словарь целиком

```
del dict_2
print(dict_2)
```

*Рисунок.5.3 – Удаление словаря с помощью ключевого слово **del***

```
    print(dict_2)
NameError: name 'dict_2' is not defined


Process finished with exit code 1
```

Рисунок.5.3.1 – Результат Рис.5.3

Методы словарей

Для словарей в Python доступны различные встроенные методы. Мы уже рассмотрели некоторые из них ранее.

dict.fromkeys(seq, value) – возвращает новый словарь с ключами из **seq** и значением, равным **value** (по умолчанию **None**)

Значение требуемого параметра **seq** – итерируемые объекты (последовательность). Оно отвечает за ключи нового словаря. Значение для параметра **value** указывать необязательно. Оно отвечает за значение по умолчанию для всех ключей. По умолчанию – **None**

```
>>> init_keys = ('player_1', 'player_2', 'player_3', 'player_4', 'player_5')
>>> score = 0
>>> players = dict.fromkeys(init_keys, score)
>>> print(players)
{'player_1': 0, 'player_2': 0, 'player_3': 0, 'player_4': 0, 'player_5': 0}
```

*Рисунок.6 – Метод **dict.fromkeys(seq, value)***

```
>>> suppliers = ('Braun', 'Mike', 'Jack', 'Robert', 'Erik')  # поставщики
>>> company_name = dict.fromkeys(suppliers)  # название компании
>>> print(company_name)
{'Braun': None, 'Mike': None, 'Jack': None, 'Robert': None, 'Erik': None}
```

*Рисунок.6.1 – Метод **dict.fromkeys(seq)***

dict.keys() – возвращает новый объект, содержащий ключи словаря

colors.keys() – возвращает все ключи в словаре

```
>>> colors = {'white': '#ffffff', 'black': '#000000', 'red': '#ff0000', 'green': '#008000', 'blue': '#0000FF'}
>>> color_name = colors.keys()
>>> print(color_name)
dict_keys(['white', 'black', 'red', 'green', 'blue'])
```

*Рисунок.7 – Метод **dict.keys()***

dict.values() – возвращает новый объект, содержащий значения словаря

colors.values() – возвращает все значения в словаре

```
>>> colors = {'white': '#ffffff', 'black': '#000000', 'red': '#ff0000', 'green': '#008000', 'blue': '#0000FF'}
>>> color_code = colors.values()
>>> print(color_code)
dict_values(['#ffffff', '#000000', '#ff0000', '#008000', '#0000FF'])
```

*Рисунок.8 – Метод **dict.values()***

dict.items() – возвращает новый объект, содержащий элементы словаря в формате (ключ, значение)

colors.items() – возвращает список кортежей, каждый из которых является парой из ключа и значения.

```
>>> colors = {'white': '#ffffff', 'black': '#000000', 'red': '#ff0000', 'green': '#008000', 'blue': '#0000FF'}
>>> color_name_and_code = colors.items()
>>> print(color_name_and_code)
dict_items([('white', '#ffffff'), ('black', '#000000'), ('red', '#ff0000'), ('green', '#008000'), ('blue', '#0000FF')])
```

*Рисунок.9 – Метод **dict.items()***

dict.get(key, default) – возвращает значение по заданному ключу. Если ключ не существует, возвращает **default** (по умолчанию **None**)

```
>>> colors = {'white': '#ffffff', 'black': '#000000', 'red': '#ff0000', 'green': '#008000', 'blue': '#0000FF'}
>>> color_red = colors.get('red')  # ключ существует, поэтому в переменной color_red присвоится значение #ff0000
>>> print(color_red)
#ff0000
>>> color_gray = colors.get('gray', 'Не найден!')  # ключ не существует, метод get возвращает значение по умолчанию 'Не найден!'
>>> print(color_gray)
Не найден!
>>> color_indigo = colors.get('indigo')  # ключ не существует, default не указан, метод get возвращает значение по умолчанию None
>>> print(color_indigo)
None
```

*Рисунок.10 – Метод **dict.get(key, default)***

dict.setdefault(keyname, value) – возвращает соответствующее значение, если ключ находится в словаре. Если нет, вставляет ключ со значением **value** и возвращает **value** (по умолчанию None)

Этот метод используется, когда нужно получить значение элемента с конкретным ключом. Если ключ не найден, он будет вставлен в словарь вместе с указанным значением.

```
>>> colors = {'white': '#ffffff', 'black': '#000000', 'red': '#ff0000'}
>>> colors.setdefault('indigo', '#4B0082')  # ключ не существует, вставляет ключ и значение в словарь, и возвращает значение ключа
'#4B0082'
>>> print(colors)
{'white': '#ffffff', 'black': '#000000', 'red': '#ff0000', 'indigo': '#4B0082'}
>>> colors.setdefault('blue')  # ключ не существует, значение ключа не указано, по умолчанию возвращает значение None
>>> print(colors)
{'white': '#ffffff', 'black': '#000000', 'red': '#ff0000', 'indigo': '#4B0082', 'blue': None}
>>> colors.setdefault('white', 'белый')  # ключ white существует со значением, возвращается значение из словаря, 'белый' игнорируется
'#ffffff'
>>> print(colors)
{'white': '#ffffff', 'black': '#000000', 'red': '#ff0000', 'indigo': '#4B0082', 'blue': None}
```

*Рисунок.11 – Метод **dict.setdefault(keyname, value)***

dict.update(other) – обновляет словарь парами ключ-значение из **other**, перезаписывая существующие ключи

```
>>> dict_1 = {'brand': 'DELL', 'founded': 1984, 'founder': 'Michael Dell', 'city': 'USA'}
>>> dict_2 = {'brand': 'Hewlett-Packard', 'short_name': 'HP', 'founded': 1939}
>>> dict_2.update(dict_1)
>>> print(dict_2)
{'brand': 'DELL', 'short_name': 'HP', 'founded': 1984, 'founder': 'Michael Dell', 'city': 'USA'}
```

*Рисунок.12 – Метод **dict.update(other)***

Этот метод устарел, вместо него можно использовать оператор Update **|=** (доступен начиная с Python 3.9), см. рис. 4.4

dict.pop(key, default) – удаляет элемент по ключу и возвращает его значение или **default**, если ключ не найден. Если **default** не указан и ключ не найден, возникает **KeyError**

```
>>> colors = {'white': '#ffffff', 'black': '#000000', 'red': '#ff0000'}
>>> elem_1 = colors.pop('white')  # ключ существует, поэтому удаляет элемент и возвращает его значение
>>> print(elem_1)
#ffffff
>>> print(colors)
{'black': '#000000', 'red': '#ff0000'}
>>> elem_2 = colors.pop('orange', '#ffa500')  # ключ не существует, по умолчанию возвращает переданное значение 'ffa500'
>>> print(elem_2)
#ffa500
>>> elem_3 = colors.pop('light_green')  # ключ не существует, значение по умолчанию не передается, возникает KeyError
Traceback (most recent call last):
  File "<input>", line 1, in <module>
KeyError: 'light_green'
```

*Рисунок.13 – Метод **dict.pop(key, default)***

dict.popitem() – удаляет и возвращает элемент (пару ключ, значение), добавленный в словарь последним. До Python 3.7 удалялся и возвращался произвольный элемент. Вызывает **KeyError**, если словарь пуст

```
>>> colors = {'red': '#ff0000', 'indigo': '#4B0082'}
>>> last_elem_1 = colors.popitem()
>>> print(last_elem_1)
('indigo', '#4B0082')
>>> print(colors)
{'red': '#ff0000'}
>>> last_elem_2 = colors.popitem()
>>> print(last_elem_2)
('red', '#ff0000')
>>> print(colors)
{}
>>> elem = colors.popitem()
Traceback (most recent call last):
  File "<input>", line 1, in <module>
KeyError: 'popitem(): dictionary is empty'
```

*Рисунок.14 – Метод **dict.popitem()***

dict.copy() – возвращает копию словаря

Была создана копия словаря *colors*. Она присвоена переменной *temp*. Если вывести *temp* в консоль, то в ней будут те же элементы, что и в словаре *colors*.

Это удобно, потому что изменения в скопированном словаре не затрагивают оригинальный словарь.

```
>>> colors = {'yellow': '#ffff00', 'indigo': '#4B0082'}
>>> temp = colors.copy()
>>> print(temp)
{'yellow': '#ffff00', 'indigo': '#4B0082'}
>>> last_elem = temp.popitem()
>>> print(temp)
{'yellow': '#ffff00'}
>>> print(colors)
{'yellow': '#ffff00', 'indigo': '#4B0082'}
```

*Рисунок.15 – Метод **dict.copy()***

dict.clear() – удаляет все элементы из словаря

```
>>> colors = {'yellow': '#ffff00', 'indigo': '#4B0082'}
>>> print(colors)
{'yellow': '#ffff00', 'indigo': '#4B0082'}
>>> colors.clear()
>>> print(colors)
{}
```

*Рисунок.16 – Метод **dict.clear()***

Встроенные функции

sorted() – сортирует элементы заданного итерируемого объекта в определенном порядке (по возрастанию или по убыванию) и **возвращает их в виде списка**

```
>>> alphabet = {'c': 'си','a': 'эй', 'b': 'би'}
>>> sort_alphabet = sorted(alphabet)  # сортирует ключи по возрастанию
>>> print(sort_alphabet)
['a', 'b', 'c']
>>> sort_alphabet_2 = sorted(alphabet, reverse=True)  # сортирует ключи по убыванию
>>> print(sort_alphabet_2)
['c', 'b', 'a']
```

*Рисунок.17 – Функция **sorted()***

len() – возвращает количество пар ключ-значение, имеющихся в заданном словаре

```
>>> colors = {'white': '#ffffff', 'black': '#000000', 'red': '#ff0000', 'indigo': '#4B0082'}
>>> length = len(colors)
>>> print(length)
4
```

*Рисунок.18 – Функция **len()***

min() – получить ключ с минимальным значением

```
>>> alphabet = {'c': 'си','a': 'эй', 'b': 'би'}
>>> min(alphabet)
'a'
>>> nums = {6: 'шесть', 2: 'два', 9: 'девять', 5: 'пять'}
>>> min(nums)
2
>>> mix = {'c': 'си','a': 'эй', 'b': 'би', 6: 'шесть', 2: 'два', 9: 'девять', 5: 'пять'}
>>> min(mix)
Traceback (most recent call last):
  File "<input>", line 1, in <module>
TypeError: '<' not supported between instances of 'int' and 'str'
```

*Рисунок.19 – Функция **min()***

max() – получить ключ с максимальным значением

```
>>> nums = {6: 'шесть', 2: 'два', 9: 'девять', 5: 'пять'}
>>> max(nums)
9
>>> alphabet = {'c': 'си','a': 'эй', 'b': 'би'}
>>> max(alphabet)
'c'
>>> mix = {'c': 'си','a': 'эй', 'b': 'би', 6: 'шесть', 2: 'два', 9: 'девять', 5: 'пять'}
>>> max(mix)
Traceback (most recent call last):
  File "<input>", line 1, in <module>
TypeError: '>' not supported between instances of 'int' and 'str'
```

*Рисунок.20 – Функция **max()***

Генератор словарей

Генератор словарей, как и в случае со списками, позволяет превратить один словарь в другой. В процессе этого превращения, элементы исходного словаря могут быть изменены или отобраны на основе какого-либо условия.

Основной целью использования как *list*, так и *dict comprehension* является упрощение и повышение читаемости кода.

```
>>> nums = {'one': 1, 'two': 2, 'three': 3, 'four': 4, 'five': 5}
>>> nums_power = {k: v**3 for k, v in nums.items()}  # возводим каждое значение в 3-ю степень
>>> print(nums_power)
{'one': 1, 'two': 8, 'three': 27, 'four': 64, 'five': 125}
>>> print(nums)
{'one': 1, 'two': 2, 'three': 3, 'four': 4, 'five': 5}
```

Рисунок.21 – Пример генератора словарей

```
>>> nums = {'one': 1, 'two': 2, 'three': 3, 'four': 4, 'five': 5}
>>> # сделаем символы всех ключей заглавными, используя строковый метод str.upper()
>>> nums_upper_case = {key.upper(): value for key, value in nums.items()}
>>> print(nums_upper_case)
{'ONE': 1, 'TWO': 2, 'THREE': 3, 'FOUR': 4, 'FIVE': 5}
>>> print(nums)
{'one': 1, 'two': 2, 'three': 3, 'four': 4, 'five': 5}
```

Рисунок.22 – Пример генератора словарей

```
>>> nums = {'six': 6, 'nine': 9, 'five': 5, 'eight': 8, 'two': 2, 'seven': 7, 'four': 4}
>>> nums_sort = {k: v for k, v in nums.items() if v > 4 if v < 8}  # отсортируем те пары, в которых значение больше 4, но меньше 8
>>> print(nums_sort)
{'six': 6, 'five': 5, 'seven': 7}
```

Рисунок.23 – Пример генератора словарей

Как и в случае *list comprehension*, применение условия *if-else* несколько меняет изначальную схему

```
>>> nums = {'six': 6, 'nine': 9, 'five': 5, 'eight': 8, 'two': 2, 'seven': 7, 'four': 4}
>>> # определим, какие из значений четные, а какие нечетные.
>>> even_odd = {key: ('Четное' if value % 2 == 0 else 'Нечетное') for key, value in nums.items()}
>>> print(even_odd)
{'six': 'Четное', 'nine': 'Нечетное', 'five': 'Нечетное', 'eight': 'Четное', 'two': 'Четное', 'seven': 'Нечетное', 'four': 'Четное'}
```

Рисунок.24 – Пример генератора словарей

Также *dict comprehension* можно использовать вместо метода **dict.fromkeys()**

```
>>> players = ('player_1', 'player_2', 'player_3', 'player_4')
>>> score = {key: 0 for key in players}
>>> print(score)
{'player_1': 0, 'player_2': 0, 'player_3': 0, 'player_4': 0}
```

Рисунок.25 – Пример генератора словарей

- Вложенные словари

```
students = {

    'id_10034': {
        'first name': 'Александр',
        'last name': 'Иванов',
        'patronymic': 'Владимирович',
        'age': 20,
        'group': 'FY-365'
    },

    'id_20983': {
        'first name': 'Ольга',
        'last name': 'Петрова',
        'patronymic': 'Александровна',
        'age': 19,
        'group': 'HQ-125'
    },

}
```

Рисунок.26 – Пример вложенного словаря

В данном случае ключами словаря выступают id студентов, а значениями – вложенные словари с информацией о них.

```python
# вывести все значения из словаря students
for v in students.values():
    print(v)
```

Рисунок.27 – Пример вложенного словаря

```
{'first name': 'Александр', 'last name': 'Иванов', 'patronymic': 'Владимирович', 'age': 20, 'group': 'FY-365'}
{'first name': 'Ольга', 'last name': 'Петрова', 'patronymic': 'Александровна', 'age': 19, 'group': 'HQ-125'}

Process finished with exit code 0
```

Рисунок.27.1 – Результат Рис.27

Для того чтобы вывести значение элемента вложенного словаря, воспользуемся двойным ключом.

```python
students = {

    'id_10034': {
        'first name': 'Александр',
        'last name': 'Иванов',
        'patronymic': 'Владимирович',
        'age': 20,
        'group': 'FY-365'
    },

    'id_20983': {
        'first name': 'Ольга',
        'last name': 'Петрова',
        'patronymic': 'Александровна',
        'age': 19,
        'group': 'HQ-125'
    },

}

# первый ключ - нужный нам студент, второй - элемент с информацией о нем
print(students['id_10034']['group'])
```

Рисунок.28 – Вывод значений по ключу из вложенного словаря

```
FY-365

Process finished with exit code 0
```

Рисунок.28.1 – Результат Рис.28