Методы словаря, перебор элементов словаря в цикле

Мы продолжаем изучение словарей языка Python. Это занятие начнем с ознакомления основных методов, которые есть у этой коллекции. Начнем с метода

dict.fromkeys(список[, значение по умолчанию])

который формирует словарь с ключами, указанными в списке и некоторым значением. Например, передадим методу список с кодами стран:

```
a = dict.fromkeys(["+7", "+6", "+5", "+4"])
```

в результате, получим следующий словарь:

{'+7': None, '+6': None, '+5': None, '+4': None}

Обратите внимание, все значения у ключей равны None. Это значение по умолчанию. Чтобы его изменить используется второй необязательный аргумент, например:

```
a = dict.fromkeys(["+7", "+6", "+5", "+4"], "код страны")
```

на выходе получаем словарь с этим новым указанным значением:

{'+7': 'код страны', '+6': 'код страны', '+5': 'код страны', '+4': 'код страны'}

Следующий метод `d.clear()` служит для очистки словаря, то есть, удаления всех его элементов. Далее, для создания копии словаря используется метод copy:

```
d = {True: 1, False: "Ложь",
"list": [1,2,3], 5: 5}
d2 = d.copy()
d2["list"] = [5,6,7]
print(d)
print(d2)
```

Также копию можно создавать и с помощью функции dict(), о которой мы с вами говорили на предыдущем занятии: `d2 = dict(d)`

Результат будет абсолютно таким же, что и при вызове метода copy().

Следующий метод get позволяет получать значение словаря по ключу: `d.get("list")` Его отличие от оператора `d["list"]`

в том, что при указании неверного ключа не возникает ошибки, а выдается по умолчанию значение None: `print(d.get(3))`

Это значение можно изменить, указав его вторым аргументом:

```
print( d.get(3, False) )
```

Похожий метод dict.setdefault(key[, default])

возвращает значение, ассоциированное с ключом key и если его нет, то добавляет в словарь со значением None, либо default – если оно указано:

```
d.setdefault("list")
d.setdefault(3)
```

Добавит ключ 3 со значением None. Удалим его:

```
del d[3]
d.setdefault(3, "three")
```

тогда добавится этот ключ со значением «three». То есть, этот метод способен создать новую запись, но только в том случае, если ключ отсутствует в словаре.

Следующий метод `d.pop(3)`

удаляет указанный ключ и возвращает его значение. Если в нем указывается несуществующий ключ, то возникает ошибка: `d.pop("abc")`

Но мы можем легко исправить ситуацию, если в качестве второго аргумента указать значение, возвращаемое при отсутствии ключа: `d.pop("abc", False)`

Здесь возвратится False. Если же ключ присутствует, то возвращается его значение.

Следующий метод `d.popitem()`

выполняет удаление произвольной записи из словаря. Если словарь пуст, то возникает ошибка:

```
d2 = {}
d2.popitem()
```

Следующий метод `d.keys()`

возвращает коллекцию ключей. По умолчанию цикл for обходит именно эту коллекцию, при указании словаря:

```
d = {True: 1, False: "Ложь", "list": [1,2,3], 5: 5}
for x in d:
   print(x)
```

то есть, эта запись эквивалента такой:

```
for x in d.keys():
    ...
```

Если же вместо keys записать метод values:

```
for x in d.values():
    ...
```

то обход будет происходить по значениям, то есть, метод values возвращает коллекцию из значений словаря.

```
for x in d.items():
    ...
```

Последний подобный метод items возвращает записи в виде кортежей: ключ, значение. О кортежах мы будем говорить позже, здесь лишь отмечу, что к элементу кортежа можно обратиться по индексу и вывести отдельно ключи и значения: `print(x[0], x[1])`

Или, используя синтаксис множественного присваивания: `x,y = (1, 2)`

можно записать цикл for в таком виде:

```
for key, value in d.items():
    print(key, value)
```

что гораздо удобнее и нагляднее.

Следующий метод update() позволяет обновлять словарь содержимым другого

```
d = dict(one = 1, two = 2, three = "3", four = "4")
d2 = {2: "неудовлетворительно", 3: "удовлетворительно", 'four':
"хорошо", 5: "отлично"}
```

И мы хотим первый обновить содержимым второго: `d.update(d2)`

Смотрите, ключ four был заменен строкой «хорошо», а остальные, не существующие ключи были добавлены.

Если же нам нужно создать новый словарь, который бы объединял содержимое обоих, то для этого можно воспользоваться конструкцией:

```
d3 = {**d, **d2}
```