Вы знаете, что есть два способа создать словарь. Использование фигурных скобок со списком пар **key: value**, разделенным запятыми, или конструктором **dict**. Мы узнаем о методе **fromkeys**, который создает новый словарь с указанными ключами и значениями. Это синтаксис этого метода:

dict.fromkeys(keys, value)

Параметр **keys** - это последовательность элементов, которые станут ключами нового словаря. Параметр **value** является необязательным и по умолчанию имеет значение **None**, но пользователь может указать значение для всех ключей в словаре. Посмотрите на пример ниже:

planets = {'Venus', 'Earth', 'Jupiter'}    
    
# initializing by default with None   
planets\_dict = dict.fromkeys(planets)  
print(planets\_dict)  # {'Jupiter': None, 'Venus': None, 'Earth': None}  
  
# initializing with a value  
value = 'planet'  
planets\_dict = dict.fromkeys(planets, value)  
print(planets\_dict)  # {'Earth': 'planet', 'Venus': 'planet', 'Jupiter': 'planet'}  
  
# changing the value of 'Jupiter'  
planets\_dict['Jupiter'] = "giant " + planets\_dict['Jupiter']  
print(planets\_dict)  
 # {'Earth': 'planet', 'Venus': 'planet', 'Jupiter': 'giant planet'}

Слово успешно добавлено! Но теперь мы хотим создать словарь, в котором будут храниться названия спутников этих планет. У некоторых планет есть несколько спутников, у некоторых их вообще нет, поэтому в качестве значения удобнее использовать список.

# some satellites of the Solar System  
satellites = ['Moon', 'Io','Europa']  
  
# initializing with an empty list  
planets\_dict = dict.fromkeys(planets, [])  
print(planets\_dict)  # {'Jupiter': [], 'Venus': [], 'Earth': []}

Добавим элементы из списка **satellites** к соответствующим планетам. Смо трите, вот что случилось с нашим словарем:

planets\_dict['Earth'].append(satellites[0])  
planets\_dict['Jupiter'].append(satellites[1])  
planets\_dict['Jupiter'].append(satellites[2])  
print(planets\_dict)    
# {'Jupiter': ['Moon', 'Io', 'Europa'], 'Venus': ['Moon', 'Io', 'Europa'], 'Earth': ['Moon', 'Io', 'Europa']}

Мы видим, что все элементы списка **satellites** присвоены всем планетам в нашем словаре. Это произошло потому, что метод **fromkeys** назначает один и тот же объект всем ключам. Обращаясь к различным ключам словаря **planets\_dict**, мы по-прежнему ссылаемся на один и тот же список. Отличие от предыдущего примера в том, что если мы используем изменяемые объекты (список, словарь) в качестве значений, все изменения также будут применяться к нашему словарю. Решение состоит в том, чтобы использовать понимание словаря:

planets\_dict = {key: [] for key in planets}

Более подробная информация об этой операции будет предоставлена ​​в другом разделе, посвященном операциям со словарем.

**Adding items**

Предположим, мы хотим добавить элементы в существующий словарь. Вы знаете один способ сделать это - определить новый ключ и новое значение: **existing\_dict['new key'] = 'new value'**. Но есть другой способ - использовать метод **update**. Метод обновляет словарь новыми элементами из другого словаря или итерацией пар ключ-значение.

Давайте создадим словарь и определим месяцы как ключи, а среднюю температуру за этот месяц как значения. Итак, у нас есть следующий **testable** словарь:

testable = {'September': '16°C', 'December': '-10°C'}   
another\_dictionary = {'June': '21°C'}  
  
# adding items from another dictionary  
testable.update(another\_dictionary)  
print(testable)  # {'September': '16°C', 'December': '-10°C', 'June': '21°C'}  
  
# adding a key-value pair  
testable.update(October='10°C')  
print(testable)    
# {'September': '16°C', 'December': '-10°C', 'June': '21°C', 'October': '10°C'}

Если указанный ключ уже существует в словаре, метод обновит ключ новым значением:

testable = {'September': '16°C', 'December': '-10°C'}  
testable.update(December='-20°C')  
  
print(testable)  # {'September': '16°C', 'December': '-20°C'}

**Getting and removing items**

Мы научились создавать словарь и добавлять в него элементы. Но что, если нам нужно получить какое-то значение из словаря или также удалить элемент? Следующие методы помогут вам справиться с разными задачами в зависимости от ваших потребностей.

1. Получить значение из словаря по ключу.

Как вы помните, мы можем получить доступ к значению в словаре по ключу:

testable = {}  
testable['September'] = '16°C'  
  
print(testable['September'])  # 16°C

Однако, если вы попытаетесь получить доступ к несуществующему ключу, вы получите **KeyError**:

print(testable['June'])  # throws a KeyError

Чтобы избежать **KeyError**, мы можем использовать метод **get**, который возвращает **None**, если указанный ключ отсутствует в словаре:

# 'get' method does not throw an error  
print(testable.get('September'))  # 16°C  
print(testable.get('June'))  # None

С помощью метода **get** мы также можем определить значение по умолчанию, которое будет возвращено в таком случае:

print(testable.get('June', 'no temperature'))  # no temperature

2. Удалите ключ из словаря и верните значение с помощью метода **pop**.

Если указанный ключ был найден в словаре, то метод удалит его и вернет значение:

testable = {'September': '16°C', 'December': '-10°C'}  
return\_value = testable.pop('December')  
  
print(return\_value)  # -10°C  
print(testable)  # {'September': '16°C'}

Если ключ не был найден, появится **KeyError**:

testable.pop('July')  # throws a KeyError

Чтобы избавиться от него, мы можем предоставить аргумент по умолчанию, и он вернет это значение по умолчанию:

return\_value = testable.pop('July', 'no temperature')  
print(return\_value)  #  no temperature

3. Удалите и верните последний элемент (ключ, значение), добавленный в словарь, используя метод **popitem**:

testable = {'September': '16°C', 'December': '-10°C'}  
return\_value = testable.popitem()  
  
print(return\_value)  # ('December', '-10°C')  
print(testable)  # {'September': '16°C'}

Обратите внимание, если словарь пуст, появится **KeyError**:

testable = {}       
return\_value = testable.popitem()  
# KeyError: 'popitem(): dictionary is empty'

До Python 3.7 метод **popitem** удаляет и возвращает случайный элемент из словаря, а не последний добавленный.

**Cleaning the dictionary**

Все описанные выше методы при удалении возвращают значение или элемент (ключ, значение), но иногда это не то, что нам нужно. Есть два способа удалить элемент из словаря (они ничего не возвращают) или сразу все содержимое словаря.

1. Удалите (удалите из словаря) значение по его ключу с ключевым словом **del**:

testable = {'September': '16°C', 'December': '-10°C', 'July': '23°C'}  
  
# this will remove both the key and the value from dictionary object  
del testable['September']    
print(testable)  # {'December': '-10°C', 'July': '23°C'}  
  
# throws a KeyError because there's no such key in the dictionary  
del testable['May']  
   
# throws a KeyError, as we've already deleted the object by the key  
del testable['September']  
  
# deletes the whole dictionary  
del testable

2. Удалите все элементы и верните пустой словарь, используя метод **clear**:

testable = {'September': '16°C', 'December': '-10°C', 'July': '23°C'}  
  
testable.clear()  # remove all elements  
print(testable)   # {}

**Differences in removal methods**

Вы можете спросить, а есть ли разница между **dict = {}** **и dict.clear ()?** Допустим, у нас есть другая переменная, которая относится к тому же словарю:

testable = {'December': '-10°C', 'July': '23°C'}  
another\_testable = testable

Затем **dict = {}** просто создает новый пустой словарь и присваивает его нашей переменной. Вернемся к приведенному выше примеру и назначим пустой словарь **testable**:

testable = {}  
print(testable)  # {}  
print(another\_testable)  # {'December': '-10°C', 'July': '23°C'}

**another\_testable** по-прежнему указывает на исходный словарь с теми же элементами, поэтому он не меняется.

Напротив, метод **clear** очистит словарь, а также все относящиеся к нему объекты:

testable = {'December': '-10°C', 'July': '23°C'}  
  
testable.clear()  
print(testable)  # {}  
print(another\_testable)  # {}

**Summary**

Что мы узнали в этой теме?

новый метод **fromkeys** для создания альтернативных словарей; мы также выяснили его особенности,

различные методы доступа к элементам и / или их удаления по ключу (**get, pop**), а также путем добавления порядка (**popitem**),

обнаружил, как добавлять новые элементы в словарь с помощью метода **update**,

ключевое слово **del** и как его использовать,

ознакомился с особенностями очистки словаря **(dict = {}** и **dict.clear ()).**

Если вы хотите узнать больше о словарях, не забудьте ознакомиться с документацией Python.