Введение в словари (dict). Базовые операции над словарями

Мы продолжаем курс по языку Python. На этом занятии мы с вами познакомимся с новой коллекцией данных – словарем. Давайте представим, что мы в программе хотели бы описать, следующие зависимости:
house -> дом
car -> машина
tree -> дерево
road -> дорога
river -> река

Если определить значения через список: `d = ["дом", "машина", "дерево", "`

то мы получим коллекцию, где каждое значение (русское слово) будет связано с числом – индексом. И изменить эти индексы невозможно – они создаются автоматически самим списком. А нам бы хотелось обращаться к этим значениям по английским словам. Как раз это позволяет делать словарь. Он формирует коллекцию в виде ассоциативного массива с доступом к элементу по ключу.

Для создания словаря используется следующий синтаксис:

{key1: value1, key2: value2, ..., keyN:valueN}

и он определяет неупорядоченную коллекцию данных, то есть, данные в словаре не имеют строгого порядка следования. Располагаться они могут произвольным образом, но всегда связаны с одним, строго определенным ключом.

В нашем конкретном случае словарь можно определить, так:

```
d = {"house": "дом", "car": "маши
     "tree": "дерево", "road": "д
     "river": "река"}
```

Как видите, мы можем записывать значения в несколько строчек, не обязательно все писать в одну.

После создания словаря, мы можем по ключу получать нужное нам значение. Для этого записываются квадратные скобки и в них указывается ключ:

`d["house"]` возвращается значение «дом», которое связано с этим ключом.
Если же указать не существующий ключ: `d[100]`

то получим ошибку. Разумеется, ключи в словарях всегда уникальны. Если записать два одинаковых: `d = {"house": "дом", "house": "до`

то ключ «house» будет ассоциирован с последним значением – «дом 2».

Также для определения словаря в Python существует специальная встроенная функция dict(). Ей, в качестве аргументов, через запятую перечисляются пары в формате ключ=значение: `d2 = dict(one = 1, two = 2, three`

Здесь ключи преобразовываются в строки и должны определяться как и имена переменных. Например, использовать числа уже не получится:

```
d2 = dict(1 = "one", two = 2, thr
```

Это неверное имя переменной.

Один из плюсов этой функции – возможность создать словарь из списка специального вида. Что это за вид? Например, такой:

```
lst = [[2, "неудовлетворительно"]]
```

Здесь вложенные списки состоят из двух элементов, которые функцией dict() интерпретируются как ключ и значение. Если мы сформируем словарь:

```
d3 = dict(lst)
```

то, как раз, увидим такую структуру данных. Причем, в этом случае, в качестве ключей можно уже выбирать и другие типы данных, не только строки, например, числа.

Если вызвать эту функцию без параметров: `d = dict()`

то получим пустой словарь. Это эквивалентно такой записи: `d = {}`

и, чаще всего, она используется на практике, так как короче.

Давайте теперь ответим на вопрос: а что вообще можно использовать в качестве ключей? Какие типы данных? В наших примерах это было или число, или строка. Что можно брать еще? На самом деле любые неизменяемые типы. Обратите внимание – **неизменяемые**. Например, можно написать так:

```
d[True] = "Истина"
d[False] = "Ложь"
```

Здесь ключи – булевы значения.

В результате, получим словарь: {True: 'Истина', False: 'Ложь'}

Также этот пример показывает, что присваивая словарю значение с новым ключом, оно автоматически добавляется в словарь. В результате, наш изначально пустой словарь стал содержать две записи. Если же мы существующему ключу присваиваем другое значение: `d[True] = 1`

то он просто поменяет свое значение в словаре. Вот так можно добавлять и менять значения словаря. То есть, словарь относится к изменяемым типам.

А вот если ключом указать список: `d[[1,2]] = 1`

то возникнет ошибка, т.к. список – это изменяемый объект. Вообще, чаще всего в качестве ключей используются строки или числа.

Это то, что касается ключей, а вот на значения словаря никаких ограничений не накладывается – там могут самые разные данные:

```
d = {True: 1, False: "Ложь", "lis
```

Операторы и функции работы со словарем

В заключение этого занятия рассмотрим некоторые операторы и функции работы со словарем. С помощью функции: `len(d)`

ожно определить число элементов в словаре. Оператор: `del d[True]`

выполняет удаление пары ключ=значение для указанного ключа. Если записать несуществующий ключ: `del d["abc"]`

то оператор возвращает ошибку. Поэтому перед удалением лучше проверить существует ли данный ключ в словаре с помощью оператора in: `"abc" in d`

Обратите внимание, оператор in проверяет именно наличие ключа, а не значения. Например, если добавить это значение: `d[1] = "abc"`

и повторно выполнить команду: `"abc" in d`

то увидим значение False.

Также можно делать противоположную проверку на отсутствие ключа, прописывая дополнительно оператор not: `"abc" not in d`