Python IV:
Кортежи, словари и множества

Кортеж

Кортежи (tuple) в Python похожи на списки, но являются *неизменяемыми*. Это означает, что после создания кортежа вы не можете изменять его содержимое (добавлять, удалять или изменять элементы). Кортежи удобно использовать, когда вам нужно представить неизменяемый набор данных. Как и в списках, элементы в кортеже могут быть любого типа данных, в том числе и другие кортежи.

Создать кортеж можно с помощью круглых скобок ( ) или с помощью встроенной функции tuple():

```
# Создание кортежа с помощью круглых скобок
my_tuple = (1, 2, 3)
# Создание кортежа с помощью встроенной функции tuple()
my_tuple = tuple([1, 2, 3])
```

Как работать с элементами кортежа?

Элементы кортежа можно получать с помощью оператора [ ] (квадратные скобки), указав индекс элемента. Индексация в кортежах начинается с нуля:

```
my_tuple = (1, 2, 3)
print(my_tuple[0]) # Вывод: 1
print(my_tuple[1]) # Вывод: 2
print(my_tuple[2]) # Вывод: 3
```

Кортежи поддерживают методы count() и index(). Рассмотрим их подробнее.

count() возвращает количество элементов кортежа, которые равны заданному элементу:

```
my_tuple = (1, 2, 3, 4, 5, 5, 6)
count = my_tuple.count(5)
print(count) # Вывод: 2
```

index() возвращает индекс первого вхождения заданного элемента в кортеже:

```
my_tuple = (1, 2, 3, 4, 5)
index = my_tuple.index(4)
print(index) # Вывод: 3
```

Важно отметить, что методы append, remove, insert и extend, которые присутствуют у списков, не могут.

Также стоит отметить что передавая в переменную значения так:

```
a, b = 4, 6
```

4, 6 - тут является кортежем. Здесь происходит распаковка кортежа.

Также можно использовать срезы, чтобы получить подмножество элементов кортежа:

```
print(tup3[1:3]) # ('hello', True)
```

Операции, доступные для кортежей, аналогичны операциям для списков: конкатенация, умножение, проверка на наличие элемента:

```
tup4 = tup1 + tup2
tup5 = (1, 'hello') * 3
print(tup4) # (1, 2, 3, 4, 5, 6)
print(tup5) # (1, 'hello', 1, 'hello', 1, 'hello')
print(2 in tup4) # True
```

Словарь

Словари используются для хранения и обработки пар ключ-значение. Ключи должны быть уникальными и неизменяемыми (как, например, строки или числа), а значения могут быть любого типа данных, включая другие словари.

```
dict1 = {'name': 'John', 'age': 25, 'city': 'New York'}
```

В примере мы создаем словарь, содержащий три пары "ключ-значение", где ключи это 'name', 'age' и 'city', а значения это 'John', 25 и 'New York'.

Доступ к элементам словаря осуществляется по ключу:

```
print(dict1['name']) # John
```

Если ключа нет в словаре, будет возбуждено исключение KeyError.

Добавление новых элементов в словарь происходит путем присваивания значения по новому ключу:

```
dict1['email'] = 'john@example.com'
print(dict1) # {'name': 'John', 'age': 25, 'city': 'New York',
'email': 'john@example.com'}
```

Удаление элементов из словаря можно выполнить с помощью оператора del:

```
del dict1['email']
print(dict1) # {'name': 'John', 'age': 25, 'city': 'New York'}
```

Также в Python существует множество встроенных функций и методов для работы со словарями. Некоторые из них:

len() - возвращает количество элементов в словаре.

```
dict3 = {'a': 1, 'b': 2, 'c': 3}
print(len(dict3)) # 3
```

keys() - возвращает все ключи в словаре.

```
print(dict1.keys()) # dict_keys(['name', 'age', 'city'])
```

values() - возвращает все значения в словаре.

```
print(dict1.values()) # dict_values(['John', 25, 'New York'])
```

items() - возвращает все пары "ключ-значение" в словаре.

```
print(dict1.items()) # dict_items([('name', 'John'), ('age', 25),
('city', 'New York')])
```

Словари могут быть очень полезны в программировании, например, для хранения настроек приложения, для построения индексов или для работы с данными в формате JSON. Однако следует помнить, что словари *не упорядочены*, то есть порядок элементов в словаре не гарантирован.

Начиная с версии Python 3.7, словарь (dict) является упорядоченной коллекцией. Это означает, что порядок элементов в словаре соответствует порядку их добавления в словарь. Однако, не стоит полагаться на упорядоченность словаря в Python, так как это не гарантируется в более старых версиях Python. Лучше всего использовать упорядоченные коллекции, такие как список или кортеж, если вам нужна гарантированная упорядоченность элементов.

Основные методы словарей:

clear(): удаляет все элементы из словаря.
copy(): создает копию словаря.
fromkeys(seq[, value]): создает новый словарь из переданной последовательности ключей seq с значением по умолчанию value (если он передан).
get(key[, default]): возвращает значение ключа key в словаре, если такой ключ есть, иначе возвращает default (если он передан).
items(): возвращает представление всех элементов в словаре в виде (ключ,

значение) кортежей.
keys(): возвращает представление всех ключей в словаре.
pop(key[, default]): удаляет и возвращает значение ключа key из словаря, если такой ключ есть, иначе возвращает default (если он передан).
popitem(): удаляет и возвращает последний добавленный элемент в словаре в виде (ключ, значение) кортежа.
setdefault(key[, default]): возвращает значение ключа key в словаре, если такой ключ есть, иначе устанавливает значение default (если он передан) для этого ключа и возвращает его.
update([other]): обновляет словарь, добавляя элементы из другого словаря other или добавляя новые пары ключ-значение, если они переданы как аргументы.
values(): возвращает представление всех значений в словаре.

Множества

Множество (set) в Python - это неупорядоченная коллекция уникальных элементов. Основные свойства множества:

1. Элементы множества не повторяются, каждый элемент может быть представлен только один раз;
2. Множество может содержать элементы разных типов;
3. Множество не упорядочено, поэтому доступ к элементам осуществляется через итерацию.

Создание множества осуществляется с помощью функции set() или при помощи фигурных скобок { }:

```
set1 = set([1, 2, 3, 4, 5])
set2 = {5, 6, 7, 8, 9}
print(set1) # {1, 2, 3, 4, 5}
print(set2) # {5, 6, 7, 8, 9}
```

Добавление элементов в множество осуществляется методом add():

```
set1 = {1, 2, 3}
set1.add(4)
print(set1) # {1, 2, 3, 4}
```

Удаление элементов из множества можно произвести с помощью методов remove() или discard():

```
set1 = {1, 2, 3, 4}
set1.remove(3)
print(set1) # {1, 2, 4}
set1.discard(5)
print(set1) # {1, 2, 4}
```

Операции с множествами в Python:

union() - объединение множеств:

```
set1 = {1, 2, 3}
set2 = {3, 4, 5}
set3 = set1.union(set2)
print(set3) # {1, 2, 3, 4, 5}
```

intersection() - пересечение множеств:

```
set1 = {1, 2, 3}
set2 = {3, 4, 5}
set3 = set1.intersection(set2)
print(set3) # {3}
```

difference() - разность множеств:

```
set1 = {1, 2, 3}
set2 = {3, 4, 5}
set3 = set1.difference(set2)
print(set3) # {1, 2}
```

symmetric_difference() - симметрическая разность множеств:

```
set1 = {1, 2, 3}
set2 = {3, 4, 5}
set3 = set1.symmetric_difference(set2)
print(set3) # {1, 2, 4, 5}
```

Также стоит добавить:

Можно объединить два множества с помощью оператора | или метода union(). Например: set1 = {1, 2, 3} и set2 = {3, 4, 5}. Тогда set1 | set2 или set1.union(set2) вернут множество {1, 2, 3, 4, 5}.

Можно найти элементы, которые есть в обоих множествах, с помощью оператора & или метода intersection(). Например: set1 = {1, 2, 3} и set2 = {3, 4, 5}. Тогда set1 & set2 или set1.intersection(set2) вернут множество {3}.

Можно найти элементы, которые есть в первом множестве, но не во втором, с помощью оператора - или метода difference(). Например: set1 = {1, 2, 3} и set2 = {3, 4, 5}. Тогда set1 - set2 или set1.difference(set2) вернут множество {1, 2}.

Можно найти элементы, которые есть только в одном из двух множеств, но не в обоих, с помощью оператора ^ или метода symmetric_difference(). Например: set1 = {1, 2, 3} и set2 = {3, 4, 5}. Тогда set1 ^ set2 или set1.symmetric_difference(set2) вернут множество {1, 2, 4, 5}.

Можно проверить, является ли одно множество подмножеством другого или надмножеством, с помощью методов issubset() и issuperset(). Например: set1 = {1, 2, 3} и set2 = {1, 2}. Тогда set2.issubset(set1) вернет True, а set1.issuperset(set2) также вернет True.

Можно создать копию множества с помощью метода copy(). Например: set1 = {1, 2, 3}.

Вот основные методы:

add(elem): добавляет элемент elem в множество. Если элемент уже присутствует в множестве, ничего не происходит.
clear(): удаляет все элементы из множества.
copy(): создает копию множества.
difference(other_set): возвращает новое множество, содержащее элементы, которые есть в исходном множестве, но отсутствуют в other_set.
difference_update(other_set): удаляет из множества все элементы, которые также присутствуют в other_set.
discard(elem): удаляет элемент elem из множества, если он присутствует. Если элемент не присутствует в множестве, ничего не происходит.
intersection(other_set) и intersection_update(other_set): первый метод возвращает новое множество, содержащее элементы, которые присутствуют и в исходном множестве, и в other_set, а второй удаляет из исходного множества все элементы, которые не присутствуют и в исходном множестве, и в other_set.
isdisjoint(other_set): возвращает True, если множество не имеет общих элементов с other_set, и False в противном случае.
issubset(other_set) и issuperset(other_set): первый метод возвращает True, если все элементы множества присутствуют в other_set, а второй - если все элементы other_set присутствуют в исходном множестве.
pop(): удаляет и возвращает произвольный элемент из множества. Если множество пусто, возникает исключение KeyError.
remove(elem): удаляет элемент elem из множества. Если элемент не присутствует в множестве, возникает исключение KeyError.