Множества и отображения

Хешируемые объекты

- Объект называется хешируемым, если он имеет хеш-значение (целое число), которое никогда не изменяется на протяжении его жизненного цикла и возвращается методом __hash__(), и может сравниваться с другими объектами (реализует метод __eq__()). Равные хешируемые объекты должны иметь равные хеш-значения.
- Хешируемые объекты могут быть использованы как ключи словарей и члены множеств.
- Все стандартные неизменяемые объекты хешируемые. Все стандартные изменяемые объекты не хешируемые.

Множества

- Множество – это неупорядоченная коллекция хешируемых объектов, которые не повторяются.
- Обычно используются для проверки элемента на вхождение в множество и удаление повторений элементов и выполнения таких операций, как объединение, пересечение, разница и симметрическая разница.
- В множествах нет понятия позиции элемента. Соответственно, они не поддерживают индексацию и срезы.
- Встроенные классы множеств: set (изменяемое множество), frozenset (неизменяемое множество).

Создание множеств

Создание множества:

- использование конструктора типа;
- перечисление элементов в фигурных скобках (только set);
- включение множеств (аналогично списковым включениям, только set).

```
empty_set = set()
empty_set = frozenset()

my_set = {1, 3, 2, 5}
my_set = frozenset([1, 3, 2, 5])

my_set = {x ** 3 for x in range(5)}
```

Операции с множествами

Операция	Описание
set([iterable]) frozenset([iterable])	создание множества (пустого или из элементов итерабельного объекта)
len(s)	количество элементов множества
x in s x not in s	проверка нахождения элемента в множестве
s.isdisjoint(t)	проверка того, что данное множество не имеет общих элементов с заданным
s.issubset(t) s <= t	проверка того, что все элементы множества s являются элементами множества t
s < t	проверка того, что s <= t и s != t
s.isuperset(t) s >= t	проверка того, что все элементы множества t являются элементами множества s
s > t	проверка того, что s >= t и s != t

Операции с множествами

Операция	Описание
s.union(t, …) s | t | …	создание нового множества, которое является объединением данных множеств
s.intersection(t, …) s & t & …	создание нового множества, которое является пересечением данных множеств
s.difference(t, …) s - t - …	создание нового множества, которое является разницей данных множеств
s.symmetric_difference(t) s ^ t	создание нового множества, которое является симметрической разницей данных множеств (то есть, разница объединения и пересечения множеств)
s.copy()	неполная копия множества s

Операции над множествами, которые являются методами, принимают в качестве аргументов любые итерабельные объекты. Операции над множествами, записанные в виде бинарных операций, требуют, чтобы второй операнд операции тоже был множеством, и возвращают множество того типа, которым было первое множество.

Операции с изменяемыми множествами

Операция	Описание
s.update(t, …) s |= t | …	добавить в данное множество элементы из других множеств
s.intersection_update(t, …) s &= t & …	оставить в данном множестве только те элементы, которые есть и в других множествах
s.difference_update(t, …) s -= t | …	удалить из данного множества те элементы, которые есть в других множествах
s.symmetric_difference_update(t) s ^= t	оставить или добавить в s элементы, которые есть либо в s, либо в t, но не в обоих множествах
s.add(element)	добавить новый элемент в множество
s.remove(element)	удалить элемент из множества; если такого элемента нет, возникает KeyError
s.discard(element)	удалить элемент из множества, если он в нём находится
s.pop()	удалить из множества и вернуть произвольный элемент (KeyError, если пустое)
s.clear()	удалить все элементы множества

Словари (ассоциативные массивы)

- Встроенным классом отображения является dict, который реализует такую структуру данных, как словарь, или ассоциативный массив, то есть неупорядоченную изменяемую коллекцию пар (ключ, значение), которая поддерживает произвольный доступ к её элементам по их ключам.
- Ключи словарей должны быть хешируемыми значениями.

Числовые ключи в словарях подчиняются правилам сравнения чисел. Таким образом, int(1) и float(1.0) считаются одинаковым ключом. Однако из-за того, что значения типа float сохраняются приближенно, не рекомендуется использовать их в качестве ключей.

Произвольное количество именованных параметров функции

- Подобно тому, как можно передавать в функции произвольное количество позиционных аргументов, которые сохраняются в кортеже, можно передавать произвольное количество именованных аргументов, которые сохраняются в словаре.
- Для этого перед именем данного словаря в списке формальных параметров ставится два символа **.
- Если используются оба способа передачи произвольного количества аргументов, параметр в форме **kwargs в сигнатуре функции должен идти после параметра в форме *args.
- Аналогично можно и распаковывать любые отображения в именованные параметры при вызове функции.

```
def function(*args, **kwargs):
    # type(args) == tuple
    # type(kwargs) == dict
    pass
```

Создание словарей

- Перечисление пар ключ-значение, разделённых символом двоеточия, через запятые в фигурных скобках:

```
{'John': 18, 'Mike': 30}
```

- Включения словарей (аналогично списковым включениям):

```
{key: value for key in keys for value in values}
```

- Использование конструктора класса dict:

```
dict(**kwargs)
dict(mapping, **kwargs)
dict(iterable, **kwargs)
```

Операции со словарями и другими отображениями

Операция	Описание
len(d)	Количество элементов.
d[key]	Получение значения с ключом key. Если такой ключ не существует и отображение реализует специальный метод __missing__ (self, key), то он вызывается. Если ключ не существует и метод __missing__ не определён, выбрасывается исключение KeyError.
d[key] = value	Изменить значение или создать новую пару ключ-значение, если ключ не существует.
key in d key not in d	Проверка наличия ключа в отображении.
iter(d)	То же самое, что iter(d.keys()).
clear()	Удалить все элементы словаря.
copy()	Создать неполную копию словаря.
@classmethod dict.fromkeys(sequence[, value])	Создаёт новый словарь с ключами из последовательности sequence и заданным значением (по умолчанию – None).

Операции со словарями и другими отображениями

Операция	Описание
d.get(key[, default])	Безопасное получение значения по ключу (никогда не выбрасывает KeyError). Если ключ не найден, возвращается значение default (по-умолчанию – None).
d.items()	В Python 3 возвращает объект представления словаря, соответствующий парам вида (ключ, значение). В Python 2 возвращает соответствующий список, а метод iteritems() возвращает итератор. Аналогичный метод в Python 2.7 – viewitems().
d.keys()	В Python 3 возвращает объект представления словаря, соответствующий ключам словаря. В Python 2 возвращает соответствующий список, а метод iterkeys() возвращает итератор. Аналогичный метод в Python 2.7 – viewkeys().
d.pop(key[, default])	Если ключ key существует, удаляет элемент из словаря и возвращает его значение. Если ключ не существует и задано значение default, возвращается данное значение, иначе выбрасывается исключение KeyError.
d.popitem()	удаляет произвольную пару ключ-значение и возвращает её. Если словарь пустой, возникает исключение KeyError.

Операции со словарями и другими отображениями

Операция	Описание
d.setdefault(key[, default])	Если ключ key существует, возвращает соответствующее значение. Иначе создаёт элемент с ключом key и значением default. default по умолчанию равен None.
d.update(mapping)	Принимает либо другой словарь или отображение, либо итерабельный объект, состоящий из итерабельных объектов – пар ключ-значение, либо именованные аргументы. Добавляет соответствующие элементы в словарь, перезаписывая элементы с существующими ключами.
d.values()	В Python 3 возвращает объект представления словаря, соответствующий значениям. В Python 2 возвращает соответствующий список, а метод itervalues() возвращает итератор. Аналогичный метод в Python 2.7 – viewvalues().

Объекты представления словаря

Объекты, возвращаемые методами items(), keys() и values() (viewitems(), viewkeys(), viewvalues() в Python 2.7) – это объекты представления словаря. Они предоставляют динамическое представление элементов словаря, то есть изменения данного словаря автоматически отображаются и на этих объектах.

Операции с представлениями словарей:

- iter(dictview) – получение итератора по ключам, значениям или парам ключей и значений. Все представления словарей при итерировании возвращают элементы словаря в одинаковом порядке. При попытке изменить словарь во время итерирования может возникнуть исключение RuntimeError.
- len(dictview) – количество элементов в словаре.
- x in dictview – проверка существования ключа, значения или пары ключ-значение в словаре.

Последовательности

Понятие последовательности

- Последовательностью в Python называется итерабельный объект, который поддерживает эффективный доступ к элементам с использованием целочисленных индексов через специальный метод __getitem__() и поддерживает метод __len__(), который возвращает длину последовательности. К основным встроенным типам последовательностей относятся list, tuple, range, str и bytes.
- Последовательности также опционально могут реализовывать методы count(), index(), __contains__() и __reversed__() и другие.

Операция	Описание
x in s, x not in s	находится ли элемент x в последовательности s
s + t	конкатенация последовательностей
s * n, n * s	конкатенация n неполных копий последовательности s
s[i]	i-й элемент последовательности s
s[i:j], s[i:j:k]	срез последовательности s от i до j с шагом k
len(s)	длина последовательности
min(s)	минимальный элемент последовательности
max(s)	максимальный элемент последовательности
s.index(x[, i[, j]])	индекс первого вхождения x (опционально – начиная с позиции i и до позиции j)
s.count(x)	общее количество вхождений x в s
sum(s)	sum(s) – сумма элементов последовательности

Операция	Описание
s[i] = x	элемент с индексом i заменяется на x
s[i:j] = t, s[i:j:k] = t	элементы с индексами от i до j (с шагом k) заменяются содержимым итерабельного объекта t
del s[i:j], del s[i:j:k]	удаление соответствующих элементов из последовательности
s.append(x)	добавление x в конец последовательности
s.clear()	удаление всех элементов последовательности
s.copy()	неполная копия последовательности
s.extend(t)	добавление всех элементов итерабельного объекта в конец последовательности
s.insert(i, x)	вставка элемента x по индексу i
s.pop(), s.pop(i)	возврат значения по индексу i (по умолчанию – последний) и удаление его из последовательности
s.remove(x)	удаление первого вхождения x
s.reverse()	разворот последовательности в обратном порядке

Списки

Списки – это изменяемые последовательности, обычно используемые для хранения однотипных данных (хотя Python не запрещает хранить в них данные разных типов). Представлены классом list

Создание списков:

```
my_list = []
my_list = [0]
my_list = [1, 2, 3, 5, 9, 0]
my_list = [x ** 3 for x in range(10)]
my_list = list(range(8))
```

Операции со списками

- Поддерживают все общие для всех последовательностей операции.
- Поддерживают общие для изменяемых последовательностей операции
- Реализуют один дополнительный метод:

list.sort(self, *, key=None, reverse=None)

Он сортирует список при помощи операции “<”. Опциональный параметр key – функция от одного аргумента, которая извлекает ключ для сортировки, reverse – сортировка в обратном порядке, если он равен True.

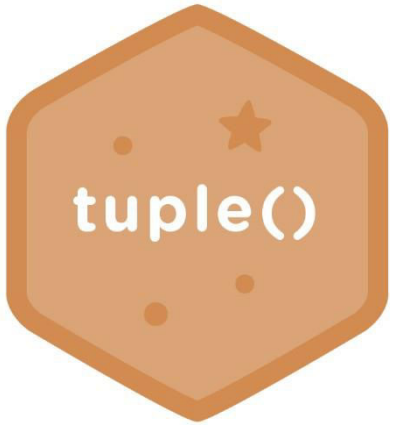

```
my_list.sort()
my_list.sort(reverse=True)
```

Кортежи

- Кортежи – это неизменяемые последовательности, обычно используемые, чтобы хранить разнотипные данные. Представлены классом tuple.
- Поддерживают все общие для последовательностей операции.

Создание кортежей:

```
my_tuple = ()
my_tuple = (1,)
my_tuple = 1,
my_tuple = (1, 2, 'a string')
my_tuple = 1, 2, 'a string'
my_tuple = tuple(range(8))
```



Распаковка кортежей

```
a, b, c = 1, 2, 3

a, b = b, a

a, b, c = iterable


head, *tail = iterable

for index, value in enumerate(sequence):
        pass
```

Функции с произвольным количеством аргументов. Распаковка аргументов функции

- Функция может иметь произвольное количество аргументов. После всех позиционных параметров функции или вместо них (но перед теми, которые предполагается использовать как именованные) в её сигнатуре можно указать специальный аргумент с символом * перед именем. Тогда оставшиеся фактические параметры сохраняются в кортеже с этим именем.
- Также существует и обратная возможность. Если при вызове функции перед именем итерабельного объекта поставить символ *, то его элементы распаковываются в позиционные аргументы.

Диапазоны

- Диапазоны – неизменяемые последовательности чисел, которые задаются началом, концом и шагом. Представлены классом range (в Python 2 – xrange; range в Python 2 – это функция, которая возвращает список).
- Начало по умолчанию равно нулю, шаг – единице. Если задать нулевой шаг, будет выброшено исключение ValueError.
- Параметры конструктора должны быть целыми числами (либо экземпляры класса int, либо любой объект с методом __index__).
- Элементы диапазона r определяются по формуле r[i] = start + step * i, где i >= 0 и r[i] < stop для step > 0 или r[i] > stop для step < 0.
- Поддерживает все общие для последовательностей операции, кроме конкатенации и повторения, а также, в версиях Python до 3.2, срезов и отрицательных индексов.

Строки

- Строки – неизменяемые последовательности кодов символов (в Python 3 – в кодировке Unicode, в Python 2 – в ASCII). Представлены классом str. В Python 2 также есть класс unicode, который представляет Unicode-строки подобно str в Python 3.
- Строковые литералы выделяются одинарными или двойными кавычками. Можно использовать утроенные кавычки для создания многострочных строк. Если перед строковым литералом стоит префикс r, то большинство escape-последовательностей игнорируются. В Python 2 префикс u задаёт Unicode-литерал.
- Поддерживают все общие для последовательностей операции, а также реализуют огромное количество собственных методов.
- Функция ord(char) возвращает код символа char, а функция chr(code) возвращает символ с кодом code.

Сравнение последовательностей

- Две последовательности равны, если они имеют одинаковый тип, равную длину и соответствующие элементы обоих последовательностей равны.
- Последовательности одинаковых типов можно сравнивать. Сравнения происходят в лексикографическом порядке: последовательность меньшей длины меньше, чем последовательность большей длины, если же их длины равны, то результат сравнения равен результату сравнения первых отличающихся элементов.