Коллекции

Списки

Список — это упорядоченный набор объектов, хранящихся в одной переменной. В отличие от массивов в других языках, у списков нет никаких ограничений на тип переменных, поэтому в них могут храниться разные объекты, в том числе и другие коллекции.

In [1]:
```
empty_list = []
empty_list = list()

none_list = [None] * 10

collections = ['list', 'tuple', 'dict', 'set']

user_data = [
    ['Elena', 4.4],
    ['Andrey', 4.2]
]
```

В питоне не нужно явно указывать размер списка или вручную выделять на него память. Длину списка можно узнать с помощью встроенной функции len. Размер списка хранится в структуре, с помощью которой реализован тип список, поэтому длина вычисляется за константное время.

In [2]:
```
len(collections)
```

Out[2]: 4

Индексы и срезы

Чтобы обратиться к конкретному элементу списка, мы используем тот же механизм, что и для строк. Нумерация элементов начинается с нуля.

In [3]:
```
print(collections)

print(collections[0])
print(collections[-1])
```

```
['list', 'tuple', 'dict', 'set']
list
set
```

Мы можем использовать доступ по индексу для присваивания.

In [4]:
```
collections[3] = 'frozenset'
print(collections)
```

```
['list', 'tuple', 'dict', 'frozenset']
```

Если попробовать обратиться к несуществующему индексу, то возникнет ошибка

In [5]:
```
collections[10]
```

```
---------------------------------------------------
-------------------------
IndexError
Traceback (most recent call last)
<ipython-input-5-218b950f302f> in <module>()
----> 1 collections[10]

IndexError: list index out of range
```

Проверить, содержит ли список некоторый объект, можно с помощью ключевого слова "in"

In [6]:
```
'tuple' in collections
```

Out[6]:
```
True
```

Срезы в списках работают точно так же, как и в строках. Создадим список из 10 элементов с помощью встроенной функции range.

In [7]:
```
range_list = list(range(10))
print(range_list)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

In [8]:
```
range_list[1:3]
```

Out[8]:
```
[1, 2]
```

In [9]:
```
range_list[3:]
```

Out[9]:
```
[3, 4, 5, 6, 7, 8, 9]
```

In [10]:
```
range_list[:5]
```

Out[10]: [0, 1, 2, 3, 4]

In [11]:
```
print(range_list)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

In [12]:
```
range_list[::2]
```

Out[12]: [0, 2, 4, 6, 8]

In [13]:
```
range_list[::-1]
```

Out[13]: [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]

In [14]:
```
range_list[1::2]
```

Out[14]: [1, 3, 5, 7, 9]

In [15]:
```
range_list[5:1:-1]
```

Out[15]: [5, 4, 3, 2]

In [16]:
```
range_list[:] is range_list
```

Out[16]: False

Итерация

Списки как и строки поддерживают протокол итерации

In [17]:
```
collections = ['list', 'tuple', 'dict', 'set']

for collection in collections:
    print('Learning {}...'.format(collection))
```

```
Learning list...
Learning tuple...
Learning dict...
Learning set...
```

Часто бывает нужно получить индекс текущего элемента при итерации. Для этого можно использовать встроенную функцию enumerate

In [18]:
```
for idx, collection in enumerate(collections):
    print('#{} {}'.format(idx, collection))
```

```
#0 list
#1 tuple
#2 dict
#3 set
```

Добавление и удаление элементов

Списки, в отличие от строк, являются изменяемой структурой данных, а значит мы можем добавлять элементы в существующий список.

In [19]:
```
collections.append('OrderedDict')

print(collections)
```

```
['list', 'tuple', 'dict', 'set', 'OrderedDict']
```

In [20]:
```
collections.extend(['ponyset', 'unicorndict'])

print(collections)
```

```
['list', 'tuple', 'dict', 'set', 'OrderedDict', 'p
onyset', 'unicorndict']
```

In [21]:
```
collections += [None]

print(collections)
```

```
['list', 'tuple', 'dict', 'set', 'OrderedDict', 'p
onyset', 'unicorndict', None]
```

Для удаление элемента из списка можно использовать ключевое слово del.

In [22]:
```
del collections[4]

print(collections)
```

```
['list', 'tuple', 'dict', 'set', 'ponyset', 'unico
rndict', None]
```

min, max, sum

Часто нам нужно найти минимальный, максимальный элемент в массиве или посчитать сумму всех элементов, сделать это можно с помощью встроенных функций min/max/sum.

In [23]:
```
numbers = [4, 17, 19, 9, 2, 6, 10, 13]

print(min(numbers))
print(max(numbers))
print(sum(numbers))
```

```
2
19
80
```

str.join

Часто бывает полезно преобразовать список в строку, для этого можно использовать метод str.join()

In [24]:
```
tag_list = ['python', 'course', 'coursera']

print(', '.join(tag_list))
```

```
python, course, coursera
```

Сортировка

In [25]:
```
import random


numbers = []
for _ in range(10):
    numbers.append(random.randint(1, 20))

print(numbers)
```

```
[13, 9, 10, 1, 1, 13, 14, 1, 16, 4]
```

Для сортировки списка в питоне есть два способа: стандартная функция sorted, которая возвращает новый список, полученный сортировкой исходного, и метод списка .sort(), который сортирует in-place. Для сортирвоки используется алгоритм TimSort.

In [26]:
```
print(sorted(numbers))
print(numbers)
```

```
[1, 1, 1, 4, 9, 10, 13, 13, 14, 16]
[13, 9, 10, 1, 1, 13, 14, 1, 16, 4]
```

In [27]:
```
numbers.sort()
print(numbers)
```

```
[1, 1, 1, 4, 9, 10, 13, 13, 14, 16]
```

Часто бывает нужно отсортировать список в обратном порядке

In [28]:
```
print(sorted(numbers, reverse=True))
```

```
[16, 14, 13, 13, 10, 9, 4, 1, 1, 1]
```

In [29]:
```
numbers.sort(reverse=True)
print(numbers)
```

```
[16, 14, 13, 13, 10, 9, 4, 1, 1, 1]
```

In [30]:
```
print(reversed(numbers))
```

```
<list_reverseiterator object at 0x107067e10>
```

In [31]:
```
print(list(reversed(numbers)))
```

```
[1, 1, 1, 4, 9, 10, 13, 13, 14, 16]
```

Методы

Кроме рассмотренных выше методов у списка есть и другие. Об этих методах вы можете почитать в документации или help(list).

- append
- clear
- copy
- count
- extend
- index
- insert
- pop
- remove
- reverse
- sort

Кортежи

Кортеж — по сути это неизменяемый список, который мы можем хэшировать, а значит использовать в качестве ключа в словарях, о которых мы поговорим позже.

In [32]:
```
empty_tuple = ()
empty_tuple = tuple()
```

In [33]:
```
immutables = (int, str, tuple)
```

In [34]:
```
immutables[0] = float
```

```
---------------------------------------------------
-------------------------
TypeError
Traceback (most recent call last)
<ipython-input-34-70298ebdccb5> in <module>()
----> 1 immutables[0] = float

TypeError: 'tuple' object does not support item as
signment
```

In [35]:
```
blink = ([], [])
blink[0].append(0)

print(blink)
```

```
([0], [])
```

In [36]:
```
hash(tuple())
```

Out[36]:
```
3527539
```

In [37]:
```
one_element_tuple = (1,)
guess_what = (1)

type(guess_what)
```

Out[37]:
```
int
```

Списки

- Упорядоченный **изменяемый** набор объектов
- Поддерживают индексы и срезы
- Поддерживают итерацию
- Встроенные функции и методы

Кортежи

- Упорядоченный **неизменяемый** набор объектов
- Похожи на списки с поправкой на неизменяемость
- Хэшируемы

Словари

Словари в питоне хранят данные в виде пары ключ-значение. Ключи должны быть хэшируемы и в обычном словаре хранятся без гарантии порядка.

In [38]:
```
empty_dict = {}
empty_dict = dict()

collections_map = {
    'mutable': ['list', 'set', 'dict'],
    'immutable': ['tuple', 'frozenset']
}
```

Доступ к значению по ключу осуществляется за константное время, то есть не зависит от размера словаря.

In [39]:
```
print(collections_map['immutable'])
```

```
['tuple', 'frozenset']
```

In [40]:
```
print(collections_map['irresistible'])
```

```
------------------------------------------------
------------------------
KeyError
Traceback (most recent call last)
<ipython-input-40-fae0f6f2a221> in <module>()
----> 1 print(collections_map['irresistible'])

KeyError: 'irresistible'
```

Часто бывает полезно попытаться достать значение по ключу из словаря, а в случае отсутствия ключа вернуть какое-то стандартное значение.

In [41]:
```
print(collections_map.get('irresistible', 'not found')
)
```

```
not found
```

Проверка на вхождения ключа в словарь так же осуществляется за константное время и выполняется с помощью ключевого слова in

In [42]:
```
'mutable' in collections_map
```

Out[42]:
```
True
```

Добавление и удаление элементов

Словари в питоне являются изменяемой структурой данных, а значит мы можем добавлять новые значения и удалять не нужные.

In [43]:
```
beatles_map = {
    'Paul': 'Bass',
    'John': 'Guitar',
    'George': 'Guitar',
}

print(beatles_map)
```

```
{'Paul': 'Bass', 'John': 'Guitar', 'George': 'Guit
ar'}
```

In [44]:
```
beatles_map['Ringo'] = 'Drums'

print(beatles_map)
```

```
{'Paul': 'Bass', 'John': 'Guitar', 'George': 'Guit
ar', 'Ringo': 'Drums'}
```

In [45]:
```
del beatles_map['John']

print(beatles_map)
```

```
{'Paul': 'Bass', 'George': 'Guitar', 'Ringo': 'Dru
ms'}
```

In [46]:
```
beatles_map.update({
    'John': 'Guitar'
})

print(beatles_map)
```

```
{'Paul': 'Bass', 'George': 'Guitar', 'Ringo': 'Dru
ms', 'John': 'Guitar'}
```

In [47]:
```
print(beatles_map.pop('Ringo'))

print(beatles_map)
```

```
Drums
{'Paul': 'Bass', 'George': 'Guitar', 'John': 'Guit
ar'}
```

In [48]:
```
unknown_dict = {}

print(unknown_dict.setdefault('key', 'default'))
```

```
default
```

In [49]:
```
print(unknown_dict)
```

```
{'key': 'default'}
```

In [50]:
```
print(unknown_dict.setdefault('key', 'new_default'))
```

```
default
```

Итерация

Словари как и другие коллекции поддерживает протокол итерации и по умолчанию итерация идет по ключам.

In [51]:
```
print(collections_map)

for key in collections_map:
    print(key)
```

```
{'mutable': ['list', 'set', 'dict'], 'immutable':
['tuple', 'frozenset']}
mutable
immutable
```

Для итерации по ключам и значениям одновременно используется метод словаря .items().

In [52]:
```
for key, value in collections_map.items():
    print('{} — {}'.format(key, value))
```

```
mutable — ['list', 'set', 'dict']
immutable — ['tuple', 'frozenset']
```

In [53]:
```
for value in collections_map.values():
    print(value)
```

```
['list', 'set', 'dict']
['tuple', 'frozenset']
```

OrderedDict

In [54]:
```
from collections import OrderedDict


ordered = OrderedDict()

for number in range(10):
    ordered[number] = str(number)

for key in ordered:
    print(key)
```

```
0
1
2
3
4
5
6
7
8
9
```

Словари

- Изменяемый неупорядоченный набор пар ключ-значение
- Быстрый доступ к значению по ключу
- Быстрая проверка на вхождение ключа в словарь

Множества

Множество в питоне — это неупорядоченный набор уникальных объектов. Множества изменяемы и чаще всего используются для удаление дубликатов и всевозможных проверок на вхождение.

In [55]:
```
empty_set = set()
number_set = {1, 2, 3, 3, 4, 5}

print(number_set)
```

```
{1, 2, 3, 4, 5}
```

Чтобы проверить, содержится ли объект в множестве, используется уже знакомое нам ключевое слово in. Проверка выполняется за константное время, время выполнения операции не зависит от размера множества. Это достигается засчет хэширования каждого элемента структуры по аналогии со словарями. По полученному от хэш-функции ключу и происходит поиск объекта. Таким образом во множествах могут содержаться только хэшируемые объекты.

In [56]:
```
print(2 in number_set)
```

```
True
```

Чтобы добавить элемент в множество, используется метод add. Так же множества в питоне поддерживают стандартные операции на множествами, такие как объединение, разность, пересечение и симметрическая разность.

In [57]:
```
odd_set = set()
even_set = set()

for number in range(10):
    if number % 2:
        odd_set.add(number)
    else:
        even_set.add(number)

print(odd_set)
print(even_set)
```

```
{1, 3, 5, 7, 9}
{0, 2, 4, 6, 8}
```

In [58]:
```
union_set = odd_set | even_set
union_set = odd_set.union(even_set)

print(union_set)
```

```
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
```

In [59]:
```
intersection_set = odd_set & even_set
intersection_set = odd_set.intersection(even_set)

print(intersection_set)
```

```
set()
```

In [60]:
```
difference_set = odd_set - even_set
difference_set = odd_set.difference(even_set)

print(difference_set)
```

```
{1, 3, 5, 7, 9}
```

In [61]:
```
symmetric_difference_set = odd_set ^ even_set
symmetric_difference_set = odd_set.symmetric_differenc
e(even_set)

print(symmetric_difference_set)
```

```
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
```

Для удаления конкретного элемента существует метод remove, для удаления любого можно использовать pop. Остальные методы можно посмотреть в help или документации.

In [62]:
```
even_set.remove(2)
print(even_set)
```

```
{0, 4, 6, 8}
```

In [63]:
```
even_set.pop()
```

Out[63]:
```
0
```

Также в питоне существует тип frozenset, который является неизменяемым множеством.

In [64]:
```
frozen = frozenset(['Anna', 'Elsa', 'Kristoff'])

frozen.add('Olaf')
```

```
---------------------------------------------------
-------------------------
AttributeError
Traceback (most recent call last)
<ipython-input-64-962f221e1321> in <module>()
      1 frozen = frozenset(['Anna', 'Elsa', 'Krist
off'])
      2
----> 3 frozen.add('Olaf')

AttributeError: 'frozenset' object has no attribut
e 'add'
```

Множества

- Изменяемый неупорядоченный набор уникальных объектов
- Быстрая проверка на вхождение
- Математические операции над множествами