Лекция 3. Списки, кортежи и словари

Содержание:

1. Список
2. Кортежи
3. Словари
4. Множества

Список

```
1.  numbers = [1, 2, 3, 4, 5]
```

```
1.  numbers1 = []
2.  numbers2 = list()
```

```
1.  numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9]
2.  numbers2 = list(numbers)
```

```
numbers = [1, 2, 3, 4, 5]
print(numbers[0])   # 1
print(numbers[2])   # 3
print(numbers[-3])  # 3

numbers[0] = 125  # изменяем первый элемент списка
print(numbers[0])   # 125
```

```
numbers = [5] * 6  # [5, 5, 5, 5, 5, 5]
print(numbers)
```

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9]
numbers2 = list(range(1, 10))
```

```
objects = [1, 2.6, "Hello", True]
```

Перебор элементов

```
companies = ["Microsoft", "Google", "Oracle", "Apple"]
for item in companies:
    print(item)
```

```
companies = ["Microsoft", "Google", "Oracle", "Apple"]
i = 0
while i < len(companies):
    print(companies[i])
    i += 1
```

Сравнение списков

Два списка считаются равными, если они содержат один и тот же набор элементов:

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9]
numbers2 = list(range(1,10))
if numbers == numbers2:
    print("numbers equal to numbers2")
else:
    print("numbers is not equal to numbers2")
```

Методы и функции по работе со списками

- **append(item)**: добавляет элемент item в конец списка
- **insert(index, item)**: добавляет элемент item в список по индексу index
- **remove(item)**: удаляет элемент item. Удаляется только первое вхождение элемента. Если элемент не найден, генерирует исключение ValueError
- **clear()**: удаление всех элементов из списка
- **index(item)**: возвращает индекс элемента item. Если элемент не найден, генерирует исключение ValueError
- **pop([index])**: удаляет и возвращает элемент по индексу index. Если индекс не передан, то просто удаляет последний элемент.
- **count(item)**: возвращает количество вхождений элемента item в список
- **sort([key])**: сортирует элементы. По умолчанию сортирует по возрастанию. Но с помощью параметра key мы можем передать функцию сортировки.
- **reverse()**: расставляет все элементы в списке в обратном порядке

Кроме того, Python предоставляет ряд встроенных функций для работы со списками:

- **len(list)**: возвращает длину списка
- **sorted(list, [key])**: возвращает отсортированный список
- **min(list)**: возвращает наименьший элемент списка
- **max(list)**: возвращает наибольший элемент списка

Добавление и удаление элементов

Для добавления элемента применяются методы `append()` и `insert`, а для удаления - методы `remove(),pop()` и `clear()`.

```
users = ["Tom", "Bob"]
# добавляем в конец списка
users.append("Alice")  # ["Tom", "Bob", "Alice"]
# добавляем на вторую позицию
users.insert(1, "Bill")          # ["Tom", "Bill", "Bob", "Alice"]
# получаем индекс элемента
i = users.index("Tom")
# удаляем по этому индексу
removed_item = users.pop(i)             # ["Bill", "Bob", "Alice"]
last_user = users[-1]
# удаляем последний элемент
users.remove(last_user)           # ["Bill", "Bob"]
print(users)
# удаляем все элементы
users.clear()
```

Проверка наличия элемента с помощью ключевого слова **in**:

```
companies = ["Microsoft", "Google", "Oracle", "Apple"]
item = "Oracle"  # элемент для удаления
if item in companies:
    companies.remove(item)

print(companies)
```

Подсчет вхождений, метод count():

```
users = ["Tom", "Bob", "Alice", "Tom", "Bill", "Tom"]

users_count = users.count("Tom")
print(users_count)      # 3
```

Сортировка

```
1. users = ["Tom", "Bob", "Alice", "Sam", "Bill"]
2.
3. users.sort()
4. print(users)        # ["Alice", "Bill", "Bob", "Sam", "Tom"]
```

```
1. users = ["Tom", "Bob", "Alice", "Sam", "Bill"]
2.
3. users.sort()
4. users.reverse()
5. print(users)        # ["Tom", "Sam", "Bob", "Bill", "Alice"]
```

```
sers = ["Tom", "bob", "alice", "Sam", "Bill"]

users.sort(key=str.lower)
print(users)       # ["alice", "Bill", "bob", "Sam", "Tom"]
```

```
users = ["Tom", "bob", "alice", "Sam", "Bill"]

sorted_users = sorted(users, key=str.lower)
print(sorted_users)       # ["alice", "Bill", "bob", "Sam", "Tom"]
```

Минимальное и максимальное значения

```
numbers = [9, 21, 12, 1, 3, 15, 18]
print(min(numbers))     # 1
print(max(numbers))     # 21
```

Копирование списков

При копировании списков следует учитывать, что списки представляют изменяемый (mutable) тип, поэтому если обе переменных будут указывать на один и тот же список, то изменение одной переменной, затронет и другую переменную:

```
1. users1 = ["Tom", "Bob", "Alice"]
2. users2 = users1
3. users2.append("Sam")
4. # users1 и users2 указывают на один и тот же список
5. print(users1)   # ["Tom", "Bob", "Alice", "Sam"]
6. print(users2)   # ["Tom", "Bob", "Alice", "Sam"]
```

глубокое копирование (deep copy)

```
import copy

users1 = ["Tom", "Bob", "Alice"]
users2 = copy.deepcopy(users1)
users2.append("Sam")
# пееменные users1 и users2 указывают на разные списки
print(users1)   # ["Tom", "Bob", "Alice"]
print(users2)   # ["Tom", "Bob", "Alice", "Sam"]
```

Копирование части списка

```
users = ["Tom", "Bob", "Alice", "Sam", "Tim", "Bill"]

slice_users1 = users[:3]   # с 0 по 3
print(slice_users1)   # ["Tom", "Bob", "Alice"]

slice_users2 = users[1:3]   # с 1 по 3
print(slice_users2)   # ["Bob", "Alice"]

slice_users3 = users[1:6:2]   # с 1 по 6 с шагом 2
print(slice_users3)   # ["Bob", "Sam", "Bill"]
```

Соединение списков

```
users1 = ["Tom", "Bob", "Alice"]
users2 = ["Tom", "Sam", "Tim", "Bill"]
users3 = users1 + users2
print(users3)   # ["Tom", "Bob", "Alice", "Tom", "Sam", "Tim", "Bill"]
```

Списки списков

```
users = [
    ["Tom", 29],
    ["Alice", 33],
    ["Bob", 27]
]

print(users[0])          # ["Tom", 29]
print(users[0][0])       # Tom
print(users[0][1])       # 29
```

Добавление, удаление и исменение общего списка, а также вложенных списков аналогично тому, как это делается с обычными (одномерными) списками:

```
users = [
    ["Tom", 29],
    ["Alice", 33],
    ["Bob", 27]
]
# создание вложенного списка
user = list()
user.append("Bill")
user.append(41)
# добавление вложенного списка
users.append(user)
print(users[-1])        # ["Bill", 41]
# добавление во вложенный список
users[-1].append("+79876543210")
print(users[-1])        # ["Bill", 41, "+79876543210"]
# удаление последнего элемента из вложенного списка
users[-1].pop()
print(users[-1])        # ["Bill", 41]
# удаление всего последнего вложенного списка
users.pop(-1)
# изменение первого элемента
users[0] = ["Sam", 18]
print(users)            # [ ["Sam", 18], ["Alice", 33], ["Bob", 27]]
```

Перебор вложенных списков:

```
users = [
    ["Tom", 29],
    ["Alice", 33],
    ["Bob", 27]
]

for user in users:
    for item in user:
        print(item, end=" | ")
```

Кортежи

Кортеж (tuple) представляет последовательность элементов, которая во многом похожа на список за тем исключением, что кортеж является неизменяемым (immutable) типом.

```
1.  user = ("Tom", 23)
2.  print(user)
```

```
1.  user = "Tom", 23
2.  print(user)
```

```
1.user = ("Tom",)
```

```
users_list = ["Tom", "Bob", "Kate"]
users_tuple = tuple(users_list)
print(users_tuple)      # ("Tom", "Bob", "Kate")
```

```
users = ("Tom", "Bob", "Sam", "Kate")
print(users[0])     # Tom
print(users[2])     # Sam
print(users[-1])     # Kate

# получим часть кортежа со 2 элемента по 4
print(users[1:4])       # ("Bob", "Sam", "Kate")
```

запись работать не будет: `users[1] = "Tim"`

При необходимости мы можем разложить кортеж на отдельные переменные:

```
user = ("Tom", 22, False)
name, age, isMarried = user
print(name)                 # Tom
print(age)                  # 22
print(isMarried)            # False
```

Особенно удобно использовать кортежи, когда необходимо возвратить из функции сразу несколько значений. Когда функция возвращает несколько значений, фактически она возвращает в кортеж:

```
def get_user():
    name = "Tom"
    age = 22
    is_married = False
    return name, age, is_married


user = get_user()
print(user[0])          # Tom
print(user[1])          # 22
print(user[2])          # False
```

Перебор кортежей

```
1.  user = ("Tom", 22, False)
2.  for item in user:
3.      print(item)
```

```
user = ("Tom", 22, False)

i = 0
while i < len(user):
    print(user[i])
    i += 1
```

Сложные кортежи

```
countries = (
    ("Germany", 80.2, (("Berlin",3.326), ("Hamburg", 1.718))),
    ("France", 66, (("Paris", 2.2),("Marsel", 1.6)))
)

for country in countries:
    countryName, countryPopulation, cities = country
    print("\nCountry: {}  population: {}".format(countryName, countryPopulation))
    for city in cities:
        cityName, cityPopulation = city
        print("City: {}  population: {}".format(cityName, cityPopulation))
```

Словари

```
dictionary = { ключ1:значение1, ключ2:значение2, ....}
```

```
1.  users = {1: "Tom", 2: "Bob", 3: "Bill"}
2.
3.  elements = {"Au": "Золото", "Fe": "Железо", "H": "Водород", "O":
    "Кислород"}
```

Мы можем также вообще определить пустой словарь без элементов:

```
objects = {}

objects = dict()
```

Преобразование из списка в словарь

```
users_list = [
    ["+111123455", "Tom"],
    ["+384767557", "Bob"],
    ["+958758767", "Alice"]
]
users_dict = dict(users_list)
print(users_dict)  # {"+111123455": "Tom", "+384767557": "Bob", "+958758767": "Alice"}
```

```
users_tuple = (
    ("+111123455", "Tom"),
    ("+384767557", "Bob"),
    ("+958758767", "Alice")
)
users_dict = dict(users_tuple)
print(users_dict)
```

Получение и изменение элементов

```
users = {
    "+11111111": "Tom",
    "+33333333": "Bob",
    "+55555555": "Alice"
}

# получаем элемент с ключом "+11111111"
print(users["+11111111"])      # Tom

# установка значения элемента с ключом "+33333333"
users["+33333333"] = "Bob Smith"
print(users["+33333333"])      # Bob Smith
```

Если при установки значения элемента с таким ключом в словаре не окажется, то произойдет его добавление:

```
users["+4444444"] = "Sam"
```

Но если мы попробуем получить значение с ключом, которого нет в словаре, то Python сгенерирует ошибку KeyError:

```
user = users["+4444444"]    # KeyError
```

```
key = "+55555555"
user = users.get(key)
user = users.get(key, "Unknown user")
```

Удаление

```
users = {
    "+11111111": "Tom",
    "+33333333": "Bob",
    "+55555555": "Alice"
}

del users["+55555555"]
print(users)
```

```
key = "+55555555"
if key in users:
    user = users[key]
    del users[key]
    print(user, "удален")
else:
    print("Элемент не
найден")
```

```
users = {
    "+11111111": "Tom",
    "+33333333": "Bob",
    "+55555555": "Alice"
}
key = "+55555555"
user = users.pop(key)
print(user)

user = users.pop("+4444444", "Unknown user")
print(user)
```

Копирование и объединение словарей

```
users = {"+1111111": "Tom","+3333333": "Bob","+5555555": "Alice"}
users2 = users.copy()
```

Метод **update()** объединяет два словаря:

```
users = {"+1111111": "Tom","+3333333": "Bob","+5555555": "Alice"}

users2 = {"+2222222": "Sam","+6666666": "Kate"}
users.update(users2)

print(users)    # {"+1111111": "Tom", "+3333333": "Bob", "+5555555": "Alice",
          "+2222222": "Sam", "+6666666": "Kate"}
print(users2)   # {"+2222222": "Sam", "+6666666": "Kate"}
```

Перебор словаря

```
users = {
    "+11111111": "Tom",
    "+33333333": "Bob",
    "+55555555": "Alice"
}
for key in users:
    print(key, " - ", users[key])
```

```
for key, value in users.items():
    print(key, " - ", value)
```

Комплексные словари

```
users = {
    "Tom": {
        "phone": "+971478745",
        "email": "tom12@gmail.com"
    },
    "Bob": {
        "phone": "+876390444",
        "email": "bob@gmail.com",
        "skype": "bob123"
    }
}
```

```
old_email = users["Tom"]["email"]
users["Tom"]["email"] = "supertom@gmail.com"
```

Множества

Множество (set) представляют еще один вид набора элементов. Для определения множества используются фигурные скобки, в которых перечисляются элементы:

```
users = {"Tom","Bob","Alice", "Tom"}
print(users)    # {"Tom","Bob","Alice"}
```

множество содержит только **уникальные** значения

```
users3 = set(["Mike", "Bill", "Ted"])
```

```
users = set()
```

Для получения длины множества применяется встроенная функция **len()**:

```
users = {"Tom","Bob","Alice"}
print(len(users)}   # 3
```

Добавление элементов

```
users = set()
users.add("Sam")
print(users)
```

Удаление элементов

```
users = {"Tom", "Bob", "Alice"}

user = "Tom"
if user in users:
    users.remove(user)
print(users)    # {"Bob", "Alice"}
```

```
user = "Tim"
users.discard(user)
```

```
users.clear()
```

Перебор множества

```
users = {"Tom","Bob","Alice"}

for user in users:
    print(user)
```

Операции с множествами

С помощью метода **copy()** можно скопировать содержимое одного множества в другую переменную:

```
users = {"Tom","Bob","Alice"}
users3 = users.copy()
```

Метод **union()** объединяет два множества и возвращает новое множество:

```
users = {"Tom","Bob","Alice"}
users2 = {"Sam","Kate", "Bob"}

users3 = users.union(users2)
print(users3)   # {"Bob", "Alice", "Sam", "Kate", "Tom"}
```

Пересечение множеств позволяет получить только те элементы, которые есть одновременно в обоих множествах.
Метод **intersection()** производит операцию пересечения множеств и возвращает новое множество:

```
users = {"Tom","Bob","Alice"}
users2 = {"Sam","Kate", "Bob"}

users3 = users.intersection(users2)
print(users3)   # {"Bob"}
```

Вместо метода intersection мы могли бы использовать операцию логического умножения:

```
users = {"Tom","Bob","Alice"}
users2 = {"Sam","Kate", "Bob"}

print(users & users2)   # {"Bob"}
```

Еще одна операция - разность множеств возвращает те элементы, которые есть в первом множестве, но отсутствуют во втором. Для получения разности множеств можно использовать метод **difference** или операцию вычитания:

```
users = {"Tom","Bob","Alice"}
users2 = {"Sam","Kate", "Bob"}

users3 = users.difference(users2)
print(users3)               # {"Tom", "Alice"}
print(users - users2)       # {"Tom", "Alice"}
```

Отношения между множествами

Метод **issubset** позволяет выяснить, является ли текущее множество подмножеством (то есть частью) другого множества:

```
users = {"Tom", "Bob", "Alice"}
superusers = {"Sam", "Tom", "Bob", "Alice", "Greg"}

print(users.issubset(superusers))   # True
print(superusers.issubset(users))   # False
```

Метод **issuperset**, наоборот, возвращает True, если текущее множество является надмножеством (то есть содержит) для другого множества:

```
users = {"Tom", "Bob", "Alice"}
superusers = {"Sam", "Tom", "Bob", "Alice", "Greg"}

print(users.issuperset(superusers))   # False
print(superusers.issuperset(users))   # True
```

frozen set

Тип **frozen set** является видом множеств, которое не может быть изменено. Для его создания используется функция**frozenset**:

```
users = frozenset({"Tom", "Bob", "Alice"})
```