Python

списки

list()

Пример создания списка

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

list()

Конструктор может принимать другой список

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

Обращение через индексы

```
1  numbers = [1, 2, 3, 4, 5]
2  print(numbers[0])    # 1
3  print(numbers[2])    # 3
4  print(numbers[-3])   # 3
5
6  numbers[0] = 125   # изменяем первый элемент списка
7  print(numbers[0])    # 125
```

Последовательный список чисел

range(end): создается набор чисел от 0 до числа end

range(start, end): создается набор чисел от числа start до числа end

range(start, end, step): создается набор чисел от числа start до числа end с шагом step

Последовательный список чисел

```
1  numbers = list(range(10))
2  print(numbers)       # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
3  numbers = list(range(2, 10))
4  print(numbers)       # [2, 3, 4, 5, 6, 7, 8, 9]
5  numbers = list(range(10, 2, -2))
6  print(numbers)       # [10, 8, 6, 4]
```

Одинаковые ли это команды?

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

Объекты списка

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

Перебор элементов

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

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

Сравнение списков

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

Методы и функции по работе со списками

Для управления элементами списки имеют целый ряд методов. Некоторые из них:

- 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

Методы и функции по работе со списками

Для управления элементами списки имеют целый ряд методов. Некоторые из них:

- 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

Методы и функции по работе со списками

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

```
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

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

print(companies)
```

Подсчет вхождений

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

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

Сортировка

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

users.sort()
print(users)       # ["Alice", "Bill", "Bob", "Sam", "Tom"]
```

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

users.sort()
users.reverse()
print(users)       # ["Tom", "Sam", "Bob", "Bill", "Alice"]
```

Сортировка

```
1  users = ["Tom", "bob", "alice", "Sam", "Bill"]
2
3  users.sort(key=str.lower)
4  print(users)       # ["alice", "Bill", "bob", "Sam", "Tom"]
```

Сортировка

Дополнительные методы сортировки

sorted(list): сортирует список list

sorted(list, key): сортирует список list, применяя к элементам функцию key

```
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

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

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

Копирование списков

И чтобы происходило копирование элементов, но при этом переменные указывали на разные списки, необходимо выполнить глубокое копирование (deep copy). Для этого можно использовать метод deepcopy(), который определен во встроенном модуле 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

list[:end]: через параметр end передается индекс элемента, до которого нужно копировать список

list[start:end]: параметр start указывает на индекс элемента, начиная с которого надо скопировать элементы

list[start:end:step]: параметр step указывает на шаг, через который будут копироваться элементы из списка. По умолчанию этот параметр равен 1.

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

```
1  users1 = ["Tom", "Bob", "Alice"]
2  users2 = ["Tom", "Sam", "Tim", "Bill"]
3  users3 = users1 + users2
4  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
```

Списки списков

Добавление, удаление и изменение 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
```

```python
# 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

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

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

```
Tom | 29 | Alice | 33 | Bob | 27 |
```