Список

Список - это непрерывная динамическая коллекция элементов.

Каждому элементу списка присваивается порядковый номер - его индекс.

Первый индекс равен нулю, второй - единице и так далее.

Основные операции для работы со списками - это **индексирование, срезы, добавление и удаление элементов, а также проверка на наличие элемента в последовательности.**

Создание пустого списка выглядит так:

```
empty_list = []
```

Создадим список, состоящий из нескольких чисел:

```
numbers = [40, 20, 90, 11, 5]
```

Создадим список, состоящий из строковых переменных:

```
fruits = ['Apple', 'Grape', 'Peach', 'Banan', 'Orange']
```

Мы можем создать список, состоящий из различных типов данных:

```
values = [3.14, 10, 'Hello world!', False, 'Python is the best']
```

Список из списков:

```
list_of_lists = [[2, 4, 0], [11, 2, 10], [0, 19, 27]]
```

Индексирование - операция обращения к элементу по его порядковому номеру.

```
fruits = ['Apple', 'Grape', 'Peach', 'Banan', 'Orange']
    print(fruits[0])
    print(fruits[1])
    print(fruits[4])
```

Результат:

```
>>> Apple
    >>> Grape
    >>> Orange
```

Индексирование работает и в обратную сторону, если мы обращаемся к элементу списка по отрицательному индексу.

Индекс с номером -1 дает нам доступ к последнему элементу, -2 к предпоследнему и так далее.

```
fruits=['Apple','Grape','Peach','Banan','Orange']
print(fruits[-1])
print(fruits[-2])
print(fruits[-3])
print(fruits[-4])
```

Результат:

```
>>>Orange
>>>Banan
>>>Peach
>>> Grape
```

Списки в Python являются изменяемым типом данных. Мы можем изменять содержимое каждой из ячеек:

```
fruits = ['Apple', 'Grape', 'Peach', 'Banan', 'Orange']
    fruits[0] = 'Watermelon'
    fruits[3] = 'Lemon'
    print(fruits)
```

Результат:

```
>>> ['Watermelon', 'Grape', 'Peach', 'Lemon', 'Orange']
```

Создание списка с помощью функции list()

```
letters = list('abcdef')
numbers = list(range(10))
even_numbers = list(range(0, 10, 2))
print(letters)
print(numbers)
print(even_numbers)
```

Результат:

```
>>> ['a', 'b', 'c', 'd', 'e', 'f'
>>> [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> [0, 2, 4, 6, 8]
```

Длина списка

- **количество элементов списка.**

Функция len()

```
fruits = ['Apple', 'Grape', 'Peach', 'Banan', 'Orange']
    print(len(fruits))
```

Результат:

5

```
string = 'Hello world'
    print(len(string))
```

Результат:

11

Срезы

Срезом называется некоторая подпоследовательность.

Принцип действия срезов очень прост: мы "отрезаем" кусок от исходной последовательности элемента, не меняя её при этом.

```
fruits = ['Apple', 'Grape', 'Peach', 'Banan', 'Orange']
part_of_fruits = fruits[0:3]
print(part_of_fruits)
```

Результат:

```
>>> ['Apple', 'Grape', 'Peach']
```

Синтаксис срезов

итерируемая_переменная[начальный_индекс : конечный_индекс - 1 : длина_шага]

```
fruits = ['Apple', 'Grape', 'Peach', 'Banan', 'Orange']
    print(fruits[0:1])
    # Если начальный индекс равен 0, то его можно опустить
    print(fruits[:2])
    print(fruits[:3])
    print(fruits[:4])
    print(fruits[:5])
    # Если конечный индекс равен длине списка, то его тоже можно
    опустить
    print(fruits[:len(fruits)])
    print(fruits[::])
```

Результат:

```
 >>> ['Apple']
>>> ['Apple', 'Grape']
>>> ['Apple', 'Grape', 'Peach']
>>> ['Apple', 'Grape', 'Peach', 'Banan']
>>> ['Apple', 'Grape', 'Peach', 'Banan', 'Orange']
>>> ['Apple', 'Grape', 'Peach', 'Banan', 'Orange']
>>> ['Apple', 'Grape', 'Peach', 'Banan', 'Orange']
```

Третий параметр среза - длина шага

```
fruits = ['Apple', 'Grape', 'Peach', 'Banan', 'Orange']
    print(fruits[::2])
    print(fruits[::3])
    # Длина шага тоже может быть отрицательной!
    print(fruits[::-1])
    print(fruits[4:2:-1])
    print(fruits[3:1:-1])
```

Результат:

```
>>> ['Apple', 'Peach', 'Orange']
>>> ['Apple', 'Banan']
>>> ['Orange', 'Banan', 'Peach', 'Grape', 'Apple']
>>> ['Orange', 'Banan']
>>> ['Banan', 'Peach']
```

С помощью цикла for мы можем перебирать значения и индексы наших последовательностей.

```
fruits = ['Apple', 'Grape', 'Peach', 'Banan', 'Orange']
for fruit in fruits:
    print(fruit, end=' ')
```

Результат:

```
>>> Apple Grape Peach Banan Orange
```

Операция in

С помощью in мы можем проверить наличие элемента в списке, строке и любой другой итерируемой переменной.

```
fruits = ['Apple', 'Grape', 'Peach', 'Banan', 'Orange']
if 'Apple' in fruits:
    print('В списке есть элемент Apple')
```

Результат:

```
>>> В списке есть элемент Apple
```

Методы для работы со списками

1.Метод append() - добавляет элемент в конец списка:

Мы можем передавать методу append() абсолютно любые значения.

```
# Создаем список, состоящий из четных чисел от 0 до 8 включительно
numbers = list(range(0,10,2))
# Добавляем число 200 в конец списка
numbers.append(200)
numbers.append(1)
numbers.append(2)
numbers.append(3)
print(numbers)
```

Результат:

```
>>> [0, 2, 4, 6, 8, 200, 1, 2, 3]
```

2. **Метод insert() -** добавляет элемент в список на произвольную позицию. insert() принимает в качестве первого аргумента позицию, на которую нужно вставить элемент, а вторым — сам элемент.

```
# Создадим список чисел от 0 до 9
numbers = list(range(10))
# Добавление элемента 999 на позицию с индексом 0
numbers.insert(0, 999)
print(numbers) # первый print
numbers.insert(2, 1024)
print(numbers) # второй print
numbers.insert(5, 'Засланная строка-шпион')
print(numbers) # третий print
```

Результат:

```
>>> [999, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> [999, 0, 1024, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> [999, 0, 1024, 1, 2, 'Засланная строка-шпион', 3, 4, 5, 6, 7, 8, 9]
```

3. Метод pop() - удаляет элемент из списка по его индексу

```
numbers = list(range(10))
print(numbers) # 1
# Удаляем первый элемент
numbers.pop(0)
print(numbers) # 2
numbers.pop(0)
print(numbers) # 3
numbers.pop(2)
print(numbers) # 4
# Чтобы удалить последний элемент, вызовем метод pop без аргументов
numbers.pop()
print(numbers) # 5
numbers.pop()
print(numbers) # 6
```

Результат:

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

4. **Метод remove()** – который удаляет первый найденный по значению элемент в списке.

```
all_types = [10, 'Python', 10, 3.14, 'Python', ['I', 'am', 'list']]
all_types.remove(3.14)
print(all_types) # 1
all_types.remove(10)
print(all_types) # 2
all_types.remove('Python')
print(all_types) # 3
```

Результат:

```
>>> [10, 'Python', 10, 'Python', ['I', 'am', 'list']] # 1
>>> ['Python', 10, 'Python', ['I', 'am', 'list']] # 2
>>> [10, 'Python', ['I', 'am', 'list']] # 3
```

5. **Метод count()** – считает элементы в списке.

```
numbers = [100, 100, 100, 200, 200, 500, 500, 500, 500, 500, 999]
print(numbers.count(100)) # 1
print(numbers.count(200)) # 2
print(numbers.count(500)) # 3
print(numbers.count(999)) # 4
```

Результат:

```
>>> 3 # 1
>>> 2 # 2
>>> 5 # 3
>>> 1 # 4
```

6. **Метод sort()** сортирует список по возрастанию значений его элементов.

```
numbers = [100, 2, 11, 9, 3, 1024, 567, 78]
    numbers.sort()
    print(numbers) # 1
    fruits = ['Orange', 'Grape', 'Peach', 'Banan', 'Apple']
    fruits.sort()
    print(fruits) # 2
```

Результат:

```
    >>> [2, 3, 9, 11, 78, 100, 567, 1024] # 1
    >>> ['Apple', 'Banan', 'Grape', 'Orange', 'Peach'] # 2
```

Мы можем изменять порядок сортировки с помощью параметра reverse. По умолчанию этот параметр равен False.

```
fruits = ['Orange', 'Grape', 'Peach', 'Banan', 'Apple']
fruits.sort()
print(fruits) # 1
fruits.sort(reverse=True)
print(fruits) # 2
```

Результат:

```
>>> ['Apple', 'Banan', 'Grape', 'Orange', 'Peach'] # 1
>>> ['Peach', 'Orange', 'Grape', 'Banan', 'Apple'] # 2
```

7. **Метод reverse()** – переворачивает список

```
numbers = [100, 2, 11, 9, 3, 1024, 567, 78]
    numbers.reverse()
    print(numbers) # 1
    fruits = ['Orange', 'Grape', 'Peach', 'Banan', 'Apple']
    fruits.reverse()
    print(fruits) # 2
```

Результат:

```
    >>> [78, 567, 1024, 3, 9, 11, 2, 100] # 1
    >>> ['Apple', 'Banan', 'Peach', 'Grape', 'Orange'] # 2
```

8. **Метод extend()** – объединяет списки.

Этот метод вызывается для одного списка, а в качестве аргумента ему передается другой список, extend() записывает в конец первого из них начало второго:

```
fruits = ['Banana', 'Apple', 'Grape']
vegetables = ['Tomato', 'Cucumber', 'Potato', 'Carrot']
fruits.extend(vegetables)
print(fruits)
```

Результат:

```
>>> ['Banana', 'Apple', 'Grape', 'Tomato', 'Cucumber', 'Potato', 'Carrot']
```

9. **Метод clear()** – очистка списка

```
fruits = ['Banana', 'Apple', 'Grape']
fruits.clear()
print(fruits)
```

Результат:

```
>>> []
```

10. **Метод index()** возвращает индекс элемента. Работает это так: вы передаете в качестве аргумента в index() значение элемента, а метод возвращает его индекс:

```
fruits = ['Banana', 'Apple', 'Grape']
print(fruits.index('Apple'))
print(fruits.index('Banana'))
print(fruits.index('Grape'))
```

Результат:

```
>>> 1
>>> 0
>>> 2
```

11. **Метод copy()** - копирует список и возвращает его брата-близнеца.

(Если есть список в списке, то внутренний список не скопируется.)

```
fruits = ['Banana', 'Apple', 'Grape']
new_fruits = fruits.copy()
fruits.pop()
print(fruits)
print(new_fruits)
```

Результат:

```
>>> ['Banana', 'Apple']
>>> ['Banana', 'Apple', 'Grape']
```

Рандомное число

```
from random import randint
n = randint(1, 10) # Случайное число от 1 до 10
```