IT CAREER HUB

Урок 14.

Методы списков

IT CAREER HUB

Методы списков

Для работы со списками доступен широкий набор встроенных методов. Поскольку списки — это изменяемые коллекции, помимо методов для анализа содержимого, существуют также методы, которые позволяют изменять элементы списка, добавлять новые, удалять существующие и сортировать данные.

Добавление элементов

Для добавления новых элементов в списки предусмотрены несколько методов, которые позволяют добавлять новые элементы изменяя исходную коллекцию, а не создавая новую.

Метод	Описание	Пример использования
`append(item)`	Добавляет один элемент в конец списка	`fruits.append("cherry")`
`extend(iterable)`	Добавляет элементы из другой коллекции по отдельности в конец списка	`numbers.extend([4, 5, 6])`
`insert(index, item)`	Вставляет элемент в список по указанному индексу	`fruits.insert(1, "blueberry")`

Метод append()

- **Метод append()** добавляет один элемент в конец списка. Один объект (например, строку, число или даже другой список), он будет помещён в конец списка как единое целое, даже если это другая последовательность.
- **Особенности:** Этот метод изменяет исходный список и увеличивает его длину на один.

Синтаксис:

```python
list.append(item)
```

- **item** — элемент, который добавляется в конец списка.

IT CAREER HUB

Примеры

```python
# добавление числа
numbers = [1, 2, 3]
numbers.append(4)
print(numbers)

# добавление строки
fruits = ["apple", "banana"]
fruits.append("cherry")
print(fruits)

# добавление другой коллекции в список
nested_list = [1, 2, 3]
nested_list.append([4, 5])
print(nested_list)

# добавление нескольких элементов
nested_list = [1, 2, 3]
nested_list.append(4, 5)  # вызовет TypeError, так как ожидается один элемент
```

Метод extend()

- **Метод extend()** расширяет список, добавляя в него элементы из другой коллекции (например, из списка, кортежа, строки и т.д.). В отличие от метода append(), который добавляет один элемент, extend() добавляет все элементы итерируемого объекта по отдельности в конец списка.
- **Особенности:** Метод изменяет исходный список и добавляет в него элементы один за другим.

Синтаксис:

```python
list.extend(iterable)
```

- **iterable** — любой итерируемый объект (например, список, строка, кортеж, объект range и т.д.), элементы которой добавляются в конец списка.

Примеры

```python
# расширение списка другой коллецией
numbers = [1, 2, 3]
numbers.extend([4, 5, 6])
print(numbers)

# расширение списком строк
fruits = ["apple", "banana"]
fruits.extend(["cherry"])
print(fruits)

# расширение списка строкой (символы строки добавляются по отдельности)
fruits = ["apple", "banana"]
fruits.extend("cherry")
print(fruits)

# расширение списка другим итеририруемым объектом (например, range)
numbers = [0]
numbers.extend(range(1, 5))
print(numbers)

# расширение списка несколькими элементами
numbers = [1, 2, 3]
numbers.extend(4, 5)  # вызовет TypeError, так как ожидается один элемент
```

```
# расширение списка неитерируемым объектом
numbers = [1, 2, 3]
numbers.extend(4)  # вызовет TypeError, так как int не является итерируемым объектом
```

IT CAREER HUB

Метод insert()

- **Метод insert()** вставляет элемент в список по указанному индексу. Элементы, находящиеся на позициях справа от вставки, смещаются вправо.
- Индексы начинаются с нуля, а если указать индекс, превышающий длину списка, новый элемент будет добавлен в конец списка.
- **Особенности:** В отличие от методов `append()` и `extend()`, `insert()` позволяет добавлять элемент в любое место списка, а не только в конец.

Синтаксис:

```python
list.insert(index, item)
```

- **index** — позиция, в которую будет вставлен элемент.
- **item** — элемент, который нужно вставить.

Примеры

```python
# вставка элемента в начало
fruits = ["apple", "banana"]
fruits.insert(0, "blueberry")
print(fruits)

# вставка элемента между другими
numbers = [1, 2, 3]
numbers.insert(1, 4)
print(numbers)

# вставка в конец списка с индексом, превышающего длину списка
animals = ["cat", "dog"]
animals.insert(10, "rabbit")
print(animals)

# вставка другой коллекции
nested_list = [1, 2, 3]
nested_list.insert(2, [4, 5])
```

```
print(nested_list)

# использование отрицательного индекса для вставки
letters = ['a', 'b', 'c']
letters.insert(-1, 'x')
print(letters)
# вставка в индекс -1 поместит элемент перед последним элементом, а не в конец списка

# вставка элемента без указания индекса
numbers = [1, 2, 3]
numbers.insert(5)  # вызовет TypeError, так как ожидается два аргумента: индекс и элемент
```

IT CAREER HUB

Задания для закрепления 1

1. Какой результат будет выведен при выполнении следующего кода?

```python
nested_list = [1, 2, 3]
nested_list.insert(1, "new")
print(nested_list)
```

a. `[1, "new", 2, 3]`
b. `[1, 2, "new", 3]`
c. `["new", 2, 3]`
d. `[1, "new", 3]`

Посмотреть ответ

2. Какой результат будет выведен при выполнении следующего кода?

```python
fruits = ["apple", "banana"]
fruits.extend("grape")
print(fruits)
```

a. `["apple", "banana", "grape"]`
b. `["apple", "banana", "g", "r", "a", "p", "e"]`
c. `Ошибка`
d. `["apple", "banana", ["grape"]]`

Посмотреть ответ

Удаление элементов

Для удаления элементов из списка предусмотрены несколько методов, которые позволяют удалять элементы из исходной коллекции, не создавая новую.

Метод remove()

- **Метод remove()** удаляет первый найденный элемент, который равен переданному аргументу.
- **Особенности:**
  - Если такого элемента нет, возникает ошибка `ValueError`.
  - Этот метод изменяет исходный список, удаляя только первое вхождение указанного значения.

Синтаксис:

```python
list.remove(item)
```

- **item** — значение, которое нужно удалить из списка.

Примеры

```python
# удаление первого вхождения элемента
fruits = ["apple", "banana", "cherry", "banana"]
fruits.remove("banana")
print(fruits)
# повторное удаление первого вхождения элемента
fruits.remove("banana")
print(fruits)

# попытка удалить несуществующий элемент ()
numbers = [1, 2, 3, 4, 5]
numbers.remove(10)  # вызовет ValueError
```

Метод pop()

- **Метод pop()** удаляет и возвращает элемент с указанного индекса. Если индекс не указан, по умолчанию удаляется последний элемент списка.
- **Особенности:**
    - Метод изменяет список, возвращая удалённый элемент.
    - Если указанный индекс выходит за пределы списка, возникает ошибка `IndexError`.

Синтаксис:

```python
list.pop(index)
```

- **index** (опционально) — индекс элемента, который нужно удалить.

Примеры

```python
# удаление и возврат последнего элемента
numbers = [10, 20, 30, 40]
last_item = numbers.pop()  # возвращенный элемент можно присвоить переменной
print(numbers)
print(last_item)

# удаление и возврат элемента по индексу
numbers = [10, 20, 30, 40]
second_item = numbers.pop(1)
print(numbers)
print(second_item)

# попытка удалить элемент по несуществующему индексу
numbers.pop(10)  # вызовет ValueError
```

Метод clear()

- **Метод clear()** полностью очищает список, удаляя все элементы, но оставляя сам объект списка в памяти.
- **Особенности:**
    - Метод изменяет список, делая его пустым.

Синтаксис:

```python
list.clear()
```

Пример

```python
# очистка всего списка
fruits = ["apple", "banana", "cherry"]
fruits.clear()
print(fruits)
```

IT CAREER HUB

Задания для закрепления 2

1. Какой результат будет выведен при выполнении следующего кода?

```python
fruits = ["apple", "banana", "cherry", "banana"]
fruits[1:].remove("banana")
print(fruits)
```

a. `["apple", "cherry", "banana"]`
b. `["apple", "banana", "cherry"]`
c. `['apple', 'banana', 'cherry', 'banana']`
d. `["apple", "cherry"]`

Посмотреть ответ

2. Что произойдет при попытке выполнить следующий код?

```python
numbers = [1, 2, 3, 4, 5]
numbers.remove(10)
```

a. `[1, 2, 3, 4, 5]`
b. `Ошибка`
c. `[1, 2, 3, 4]`
d. `[1, 2, 3, 5]`

Посмотреть ответ

IT CAREER HUB

Поиск и подсчет элементов

Python предоставляет несколько методов для поиска и подсчёта элементов в списках. Эти методы аналогичны методам кортежа и позволяют найти индекс первого вхождения элемента и подсчитать количество вхождений определённого элемента.

Метод index()

- **Метод index()** возвращает индекс первого вхождения указанного элемента item в списке.
- **Особенности:**
  - Возвращает индекс первого вхождения элемента в список.
  - Если элемент не найден в указанном диапазоне (или в списке), будет вызвано исключение `ValueError`.
  - Дополнительные параметры `start` и `stop` позволяют ограничить диапазон поиска.

Синтаксис:

```python
list.index(item, start, stop)
```

- **item** — элемент, который нужно найти в списке.
- **start (опционально)** — индекс, с которого начинается поиск. По умолчанию 0, то есть поиск начинается с начала списка.
- **stop (опционально)** — индекс, до которого выполняется поиск (не включительно). По умолчанию len(list), что означает поиск до конца списка.

Примеры

```python
# поиск первого вхождения элемента "banana"
fruits = ["apple", "banana", "cherry", "banana", "cherry"]
banana_index = fruits.index("banana")
print(banana_index)
```

```python
# поиск первого вхождения "banana", начиная с индекса 2
fruits = ["apple", "banana", "cherry", "banana", "cherry"]
banana_index = fruits.index("banana", 2)
print(banana_index)

# поиск "cherry" в диапазоне индексов от 1 до 4 (не включая 4)
fruits = ["cherry", "banana", "cherry", "banana", "cherry"]
banana_index = fruits.index("cherry", 1, 4)
print(banana_index)

# поиск индекса элемента, которого нет в списке
fruits = ["apple", "banana", "cherry", "banana"]
index = fruits.index("orange")  # вызовет ValueError
```

Метод count()

- **Метод count()** возвращает количество вхождений указанного элемента `item` в списке.
- **Особенности:** Если элементы не найдены, метод возвращает 0.

Синтаксис:

```python
list.count(item)
```

- **item** — элемент, который нужно посчитать.

Примеры

```python
# подсчёт вхождений элемента
```

```python
fruits = ["apple", "banana", "cherry", "banana"]
banana_count = fruits.count("banana")
print(banana_count)

# подсчёт элементов, которых нет в списке
orange_count = fruits.count("orange")
print(orange_count)
```

Метод sort

Метод `sort()` используется для сортировки элементов исходного списка. Его элементы будут упорядочены по возрастанию или убыванию. Метод поддерживает различные параметры, такие как выбор направления сортировки и использование функций для выбора логики сортировки.

Синтаксис:

```python
list.sort(key=None, reverse=False)
```

- **key** (опциональный параметр) — функция, которая используется для извлечения ключа для каждого элемента. По этому ключу будет происходить сравнение элементов. По умолчанию используется естественное сравнение элементов.
- **reverse** (опциональный параметр) — если `True`, элементы будут отсортированы в порядке убывания. По умолчанию `reverse=False`, что означает сортировку по возрастанию.

IT CAREER HUB

Примеры

Сортировка по возрастанию (по умолчанию):

```python
numbers = [4, 1, 7, 2, 9]
numbers.sort()
print(numbers)
```

Сортировка по убыванию с параметром `reverse=True`:

```python
numbers = [4, 1, 7, 2, 9]
numbers.sort(reverse=True)
print(numbers)
```

Сортировка строк по алфавиту:

```python
fruits = ["banana", "apple", "cherry"]
fruits.sort()  # По умолчанию лексикографическое сравнение
print(fruits)
```

Сортировка с использованием ключа (например, сортировка по длине строк):

- Параметр key позволяет задать функцию, которая извлекает ключ сортировки для каждого элемента. Нужно передавать незапущенную функцию без скобок (). Функция будет запускаться для каждого элемента коллекции.

```python
# Использование встроенной функции len()**`len()`** для сортировки строк по их длине, а не лексикографически.
```

```
fruits = ["banana", "apple", "cherry", "blueberry"]
fruits.sort(key=len)
print(fruits)

# Использование встроенной функции max() для сортировки кортежей по
максимальному элементу.
tuples = [(3, 6), (1, 7, 9), (12, 5), (1, 3, 7)]
tuples.sort(key=max, reverse=True)
print(tuples)
```

IT CAREER HUB

Метод reverse

Метод `reverse()` используется для разворота списка в обратном порядке и изменяет исходный список.

Синтаксис:

```python
list.reverse()
```

Примеры

```python
numbers = [4, 1, 7, 2, 9]
numbers.reverse()
print(numbers)

fruits = ["apple", "banana", "cherry"]
fruits.reverse()
print(fruits)
```

Функции sorted и reversed

Функции `sorted()` и `reversed()` используются для сортировки и разворота коллекций (например, списков, строк, кортежей), но в отличие от методов `sort()` и `reverse()`, они не изменяют исходную коллекцию. Вместо этого они возвращают новый отсортированный или развернутый итерируемый объект.

Функция sorted() — сортировка коллекции

Функция `sorted()` возвращает новый отсортированный список на основе итерируемого объекта.
Синтаксис аналогичен методу `sort()`, но первым аргументом передается коллекция для сортировки.

Синтаксис:

```python
sorted(iterable, key=None, reverse=False)
```

- **iterable** — любой итерируемый объект (список, кортеж, строка и т.д.).
- **key** (опциональный) — функция, которая используется для сортировки (например, len() для сортировки строк по длине).
- **reverse** (опциональный) — если `True`, сортировка выполняется в порядке убывания. По умолчанию `reverse=False`.

Примеры

```python
# Сортировка списка чисел
numbers = [3, 1, 4, 1, 5]
sorted_numbers = sorted(numbers)
print(sorted_numbers)
print(numbers)

# Сортировка строки
letters = "bca"
sorted_letters = sorted(letters)  # Вернет отсортированные символы по
отдельности
print(sorted_letters)
print(letters)

# Сортировка кортежа по длине строк в порядке убывания
words = ("apple", "banana", "kiwi")
sorted_words = sorted(words, key=len, reverse=True)
print(sorted_words)
print(words)
```

Функция reversed() — разворот коллекции

Функция `reversed()` возвращает новый итерируемый объект, представляющий элементы исходной коллекции в обратном порядке. Этот объект не является списком — это итерируемый объект, который можно преобразовать в список, строку или кортеж с помощью соответствующих функций. Исходная коллекция при этом остаётся неизменной.

Синтаксис:

```python
reversed(iterable)
```

- **iterable** — любой итерируемый объект (например, список, строка, кортеж).

Примеры

```python
# Разворот списка
numbers = [3, 1, 4, 1, 5]
reversed_numbers = reversed(numbers)  # возвращает не list, a
list_reverseiterator
print(reversed_numbers)
list_reversed_numbers = list(reversed_numbers)  # чтобы увидеть элементы, нужно
преобразовать в list
print(list_reversed_numbers)

# Разворот строки
word = "hello"
reversed_letters = reversed(word)
print(reversed_letters)
reversed_word = ''.join(reversed_letters)
print(reversed_word)
```

Таблица отличий методов сортировки и реверса

Характеристика	sort()	sorted()	reverse()	reversed()
Изменяет исходный объект?	Да	Нет	Да	Нет
Возвращаемый результат	None	Новый отсортированный список	None	Новый итерируемый объект
Работает с типами	Списки	Любые итерируемые объекты	Списки	Любые итерируемые объекты

Функции min, max, sum

Встроенные функции `min()`, `max()` и `sum()`, можно также использовать для работы с коллекциями и другими итерируемыми объектами.

Функции min(), max()

- Функции `min()` и `max()` возвращают минимальное и максимальное значения среди элементов итерируемого объекта или среди переданных аргументов. Так же как и функции сортировки, они принимают необязательный аргумент `key`, в который передается функция для вычисления критерия сравнения.

Синтаксис

```python
min(iterable, key=func)
max(iterable, key=func)
```

- **iterable** — итерируемый объект, из которого будет возвращён минимальный элемент.
- **key=func** — (необязательный) функция, которая вычисляет критерий сравнения для элементов.

IT CAREER HUB

Примеры

```python
# Поиск минимального/максимального значения в списке
numbers = [4, 1, 7, 2, 9]
print(min(numbers))
print(max(numbers))

# Поиск минимального/максимального значения среди отдельных аргументов
print(min(3, 5, 2, 8))
print(max(3, 5, 2, 8))

# Поиск минимального/максимального символа
chars = "Python"
print(min(chars))
print(max(chars))

# Использование параметра key для поиска по длине строк
words = ["apple", "banana", "kiwi", "grape"]
print(min(words, key=len))
print(max(words, key=len))
```

Функция sum()

- Функция `sum()` возвращает сумму всех элементов в итерируемом объекте. Она принимает опциональный аргумент `start`, который добавляется к итоговой сумме (по умолчанию `start=0`).

Синтаксис:

```python
sum(iterable, start=0)
```

- **iterable** — итерируемый объект (например, список, кортеж).

- **start** (опционально) — начальное значение, к которому суммировать остальные (по умолчанию 0).

Примеры

Python

```python
# Сумма чисел в списке
numbers = [4, 1, 7, 2, 9]
total_sum = sum(numbers)
print(total_sum)

# Сумма чисел в списке с начальным значением
total_sum_with_start = sum(numbers, 1000)
print(total_sum_with_start)

# Сумма чисел в кортеже
numbers_tuple = (1, 2, 3)
total_sum_tuple = sum(numbers_tuple)
print(total_sum_tuple)
```

Ответы на задания

Задания на закрепление 1	Вернуться к заданиям
1. Результат выполнения кода	Ответ: a
2. Результат выполнения кода	Ответ: b
Задания на закрепление 2	Вернуться к заданиям
1. Результат выполнения кода	Ответ: c
2. Результат выполнения кода	Ответ: b

IT CAREER HUB

IT CAREER HUB

Практическая работа

1. Начало равно концу

Напишите программу, которая принимает список строк и создает новый список только из слов,

начинающихся и заканчиваются одной и той же буквой.

Данные:

```python
strings = ["apple", "banana", "level", "radar", "grape"]
```

Пример вывода:

```python
Строки, которые начинаются и заканчиваются одной и той же буквой: ['level', 'radar']
```

Решение:

```python
strings = ["apple", "banana", "level", "radar", "grape"]
new_strings = []

for string in strings:
    if string[0] == string[-1]:
        new_strings.append(string)


print("Строки, которые начинаются и заканчиваются одной и той же буквой:",
strings)
```

IT CAREER HUB

2. Слова с гласных

Напишите программу, которая обрабатывает список строки создает новый список, где строки, начинающиеся с согласной буквы будут заканчиваться символом "!".

Остальные слова добавьте в том же виде.

Данные:

```python
strings = ["apple", "banana", "cherry", "grape", "orange"]
```

Пример вывода:

```python
Результат: ['apple!', 'banana', 'cherry', 'grape', 'orange!']
```

Решение:

```python
strings = ["apple", "banana", "cherry", "grape", "orange"]
new_strings = []

for string in strings:
    if string[0].lower() in "aeiou":
        string += "!"
    new_strings.append(string)
print("Результат:", new_strings)
```