

Методи і функції опрацювання списків

Методи

1. Метод `append()`. Додавання елемента до списку

Метод `append ()` використовується для додавання елемента до списку. Метод може отримувати тільки один параметр. Параметром методу може бути будь-який об'єкт: число, рядок, список і т.д.

Приклад використання методу `append ()`

Приклад 1

```
# Метод append () - додавання елементів до списку
```

```
# Заданий список
```

```
A = [2, 3.78, 'abcde', True]
```

```
# Додати 1 елемент до списку
```

```
A.append(7) # A = [2, 3.78, 'abcde', True, 7]
```

```
print( "A =", A)
```

```
# Сформувати список квадратів чисел від 1 до 10
```

```
# За допомогою методу append ()
```

```
B = []
```

```
i = 1;
```

```
while i <= 10:
```

```
    B.append(i * i)
```

```
    i = i + 1
```

```
print ("B =", B)
```

Результат виконання програми

```
A = [2, 3.78, 'abcde', True, 7]
```

```
B = [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

2. Метод `extend ()`. розширення списку

Даний метод дозволяє розширити список. Вхідним параметром методу є інший список, який додається до даних. Список розширення може бути іменованим об'єктом або списком, взятим у квадратні дужки `[]`.

Приклад програми, яка містить метод `extend ()`

Приклад 2

```
# Метод extend () - розширення списку
```

```
# Задані два списки
```

```
A = [2, 3.78, 'abcde', True]
```

```
B = ["Hello", 77, 1.84]
```

```
# Розширити список A на величину списку B
```

```
A.extend(B) # A = [2, 3.78, 'abcde', True, 'Hello', 77, 1.84]
```

```
print("A =", A)
```

```
# Сформуванати список квадратів чисел від 1 до 10
# За допомогою методу extend ()
D = []
i = 1;
while i <= 10:
    D.extend ([i * i]) # додати список [i * i] до списку D
    i = i + 1
print("D =", D)
```

Результат виконання програми

```
A = [2, 3.78, 'abcde', True, 'Hello', 77, 1.84]
D = [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

3. Метод insert(). Вставка елемента в список в заданій позиції

Метод insert () дозволяє вставити новий елемент в список із заданої позиції. Метод отримує два параметри. Перший параметр - позиція вставки, яка починається з 0. Другий параметр - ім'я об'єкта (значення), який вставляється.

Приклад використання методу insert ()

Приклад 3

Метод insert () - вставка одиночного елемента в список

Заданий список

```
A = [1, 2, 3, 4, 5]
```

Вставка в позицію 2 нового числа 777

```
A.insert (2, 777)
```

```
print("A =", A)
```

Сформуванати список квадратів чисел від 1 до 10

За допомогою методу insert ()

```
D = []
```

```
i = 10;
```

```
while i >= 1:
```

```
    D.insert (0, i * i)
```

```
    i = i-1
```

```
print("D =", D)
```

Результат виконання програми

```
A = [1, 2, 777, 3, 4, 5]
D = [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Метод insert () дозволяє вставляти список в список. Наприклад, якщо в вищенаведеної програмі рядок

```
A.insert(2, 777)
```

замінити на рядок

```
A.insert (2, [777])
```

то в цьому випадку, список А буде наступним

```
A = [1, 2, [777], 3, 4, 5]
```

тобто, це є звичайне використання вкладених списків.

4. Метод `index ()`. Визначення індексу елемента в списку

Метод `index ()` дозволяє отримати значення індексу (позиції) заданого елемента списку. Метод отримує 1 параметр, який є шуканим елементом. Значення індексу, який відповідає першому елементу списку, дорівнює 0.

Приклад використання методу `index ()`

Приклад 4

```
# Метод index ()
# Заданий список
A = [ 'a', 'b', 'c', 'd', 'e', 'f']
t = A.index('c')          # t = 'c'
print( "t =", t)
```

Результат виконання програми

```
t = 2
```

Якщо елемента немає в списку, то видається повідомлення про помилку. Наприклад, якщо в наведеному вище коді рядок

```
t = A.index('c')
замінити рядком
```

```
t = A.index ('g')          # t = 'c'
```

то інтерпретатор Python видасть повідомлення про помилку

```
ValueError: 'g' is not in list
```

5. Метод `count ()`. Визначення кількості входжень заданого елемента в списку

Метод `count ()` повертає кількість входжень заданого елемента в списку. Метод отримує один параметр.

Приклад 5 # Метод `count ()` - кількість входжень заданого елемента в списку

```
# Заданий список
A = [ 'a', 'b', 'c', 'd', 'e', 'f']
```

```
na = A.count('d')          # na = 1
```

```
B = [1, 3, 5, 3, 2, 4]
```

```
nb = B.count(3)           # nb = 2
```

```
print("na =", na)
print("nb =", nb)
```

Результат виконання програми

```
na = 1
nb = 2
```

6. Метод `sort ()`. Сортування списку

Метод `sort ()` використовується для сортування списку. За замовчуванням метод сортує елементи списку в порядку зростання значень. Метод може змінити порядок сортування за допомогою наступних іменованих аргументів:

`key` - аргумент, який дозволяє визначити власну функцію порівняння при виклику методу `sort ()`. Ця функція отримує один єдиний аргумент і повертає значення, яке буде використовуватися в операції порівняння;

`reverse` - аргумент, який використовується для вказівки порядку сортування елементів. Якщо `reverse = True`, то елементи списку сортуються в порядку спадання.

Приклад 1. Використання методу `sort ()` для сортування списку в порядку зростання.

```
# Приклад 6 # Метод sort () - сортування списку
```

```
# Заданий список
```

```
A = [ 'a', 'f', 'v', 'd', 'n', 'b' ]
```

```
# Сортування списку
```

```
A.sort()
```

```
B = [1, 3, 5, 10, 2, 8]
```

```
B.sort()
```

```
print("A =", A)
```

```
print("B =", B)
```

Виконання програми дає наступний результат

```
A = [ 'a', 'b', 'd', 'f', 'n', 'v' ]
```

```
B = [1, 2, 3, 5, 8, 10]
```

Для того, щоб використовувати метод `sort ()` всі елементи списку повинні бути або числовими або малими. Наприклад, наступний код

```
# Помилка!
```

```
C = ["Hello", "ABC", 7]
```

```
C.sort()
```

згенерує помилку

```
TypeError: '<' not supported between instances of 'int' and 'str'
```

Приклад 2. Сортування списку в порядку спадання. Для того, щоб список впорядкувати в порядку убутання, потрібно, щоб значення іменованого аргументу `reverse` було рівним `True`. Наведений нижче приклад сортує список `C` в порядку убутання елементів

```
# Метод sort () - сортування списку
```

```
# Заданий список
C = [2, 3, 1, 5]
C.sort(reverse = True) # впорядкувати в порядку убавання
print("C =", C)
```

Результат виконання програми

```
C = [5, 3, 2, 1]
```

Приклад 3. Сортування списку з заданим ключем key. У прикладі упорядковано рядки, які попередньо наводяться до верхнього регістру: 'aBc' => 'ABC' функцією upper ()

```
# Приклад 7 # Метод sort () - сортування списку з заданим ключем
key
# Заданий список рядків
S = ["aBc", "ABCD", "ab", "ABCC", "DEFf"]

S2 = list(S) # створити новий список
S2.sort(key = str.upper) # відсортувати за ключем key

S3 = list(S) # ще один список
S3.sort(key = str.upper, reverse = True) # відсортувати за
аргументами key і reverse

print("S=", S)
print("S2=", S2)
print("S3=", S3)
```

Результат виконання програми

```
S = [ 'aBc', 'ABCD', 'ab', 'ABCC', 'DEFf' ]
S2 = [ 'ab', 'aBc', 'ABCC', 'ABCD', 'DEFf' ]

S3 = [ 'DEFf', 'ABCD', 'ABCC', 'aBc', 'ab' ]
```

7. Метод reverse (). Реверсування списку

Метод reverse () використовується для зміни порядку проходження елементів списку на зворотний.

Приклад використання методу

```
# Приклад 8
# Метод reverse() - реверсування списку
# Задані два списки
A = [1, 2, 3, 4, 5]
B = [True, 7.78, 2.85, -1000, "bestprog.net"]

# Реверсування списків
A.reverse()
B.reverse()
```

```
print("A =", A)
print("B =", B)
```

Результат виконання програми

```
A = [5, 4, 3, 2, 1]
B = [ 'bestprog.net', -1000, 2.85, 7.78, True]
```

8. Метод pop (). Витягування елемента зі списку

Метод pop () призначений для витягування (видалення) елемента зі списку. Метод має дві реалізації, які відрізняються кількістю одержуваних параметрів:

- реалізація без параметрів. В цьому випадку витягується останній елемент списку;
- реалізація з одним параметром. У цьому випадку параметр є індексом елемента, який потрібно витягнути зі списку. Першому елементу відповідає індекс 0.

Приклад використання методу pop ()

```
# Приклад 9 # Метод pop() - зменшення списку
# Заданий список
A = [5, 3.8, True, False, "ABCD"]

# Видалити останній елемент
A.pop()          # A = [5, 3.8, True, False]

print("A =", A)

# Видалити елемент з індексом 1
A.pop(1)         # A = [5, True, False]

print ( "A =", A)
```

Результат роботи програми

```
A = [5, 3.8, True, False]
A = [5, True, False]
```

9. Метод remove (). Видалення заданого елемента зі списку

Метод remove () видаляє заданий елемент зі списку. Якщо в списку є кілька елементів із зазначеним значенням, то віддаляється перше входження заданого елемента.

Приклад використання методу remove ()

```
# Приклад 10
# Метод remove ()
# Заданий список
A = [5, 3.8, True, 3.8, True, False, "ABCD"]

# Видалити перший елемент, який дорівнює True
```

```
A.remove(True)

# Видалити перший елемент, який дорівнює 3.8
A.remove(3.8)

print ( "A =", A)
```

Результат роботи програми

```
A = [5, 3.8, True, False, 'ABCD']
```

=====

Функції

Функції для роботи зі списками, на відміну від методів, не змінюють сам список, а повертають певне значення. З функціями `len()` і `list()` ви вже знайомі.

```
# max(list) Повертає найбільше значення елемента
>>> a = [1, 5, 7, 31, -5]
>>> max(a)
31
```

```
# min(list) Повертає найменше значення елемента
>>> min(a)
-5
```

```
# sum(list) Повертає значення суми елементів
>>> sum(a)
39
```

```
# del(list[n]) Видаляє елемент із індексом n
>>> del(a[2])
>>> a
[1, 5, 31, -5]
```

Знайдемо індекс найбільшого елемента у списку `a = [1, 5, 7, 31, -5]`:

```
>>> a = [1, 5, 7, 31, -5]
>>> m = max(a)           # m = 31
>>> n = a.index(m)      # n = 3
```

Комбінуючи функції та методи роботи зі списками, ми можемо, як із цеглинок, скласти алгоритм розв'язування складної задачі.

=====