Методы списков и строк. Вложенные списки

Методы списков

Список — один из часто используемых и универсальных типов данных.

- **list.append(x)**

x : Элемент, который требуется добавить в список.

```
my_list = []
my_list.append(1)
 my_list # [1]
my_list.append(3)
 my_list # [1, 3]
```

Методы списков

list.clear([i])

Удаляет из списка все имеющиеся в нём значения.

```
my_list = [1, 2, 3]
my_list.clear() # None
 my_list # []
```

Методы списков

list.copy()

Возвращает копию списка.

Внимание: Возвращаемая копия является поверхностной (без рекурсивного копирования вложенных элементов).

```
my_list = [1, 2, 3]
my_list_copy = my_list.copy() # [1, 2, 3]
```

Методы списков

list.count(x)

```
random_list = [4, 1, 5, 4, 10, 4]
random_list.count(4) # 3
```

Методы списков

list.extend(iterable)

Дополняет список элементами из указанного объекта.

it : Объект, поддерживающий итерирование, элементами которого требуется дополнить список.

my_list = [] my_list.extend([1, 2, 3]) # None my_list # [1, 2, 3]

my_list.extend('add') # None

my_list # [1, 2, 3, 'a', 'd', 'd']

Для добавления единственного элемента используйте append().

Методы списков

list.insert(i, x)

Вставляет указанный элемент перед указанным индексом

i : Позиция (индекс), перед которой требуется поместить элемент. Нумерация ведётся с нуля. Поддерживается отрицательная индексация.

x : Элемент, который требуется поместить в список.

Метод модифицирует

```
my_list = [1, 3]
my_list.insert(1, 2)
my_list # [1, 2, 3]
my_list.insert(-1, 4)
 my_list # [1, 2, 4, 3]
```

Методы списков

list.pop([i])

Возвращает элемент [на указанной позиции], удаляя его из списка.

i=None : Позиция искомого элемента в списке (целое число). Если не указана, считается что имеется в виду последний элемент списка. Отрицательные числа поддерживаются.

```
my_list = [1, 2, 3, 4, 5]
 last = my_list.pop() # 5
my_list # [1, 2, 3, 4]
second = my_list.pop(-3) # 2
my_list # [1, 3, 4]
 first = my_list.pop(0) # 1
my_list # [3, 4]
```

Методы списков

list.remove(x)

Удаляет из списка указанный элемент.

x : Элемент, который требуется удалить из списка. Если элемент отсутствует в списке, возбуждается ValueError.

Удаляется только первый обнаруженный в списке элемент, значение которого совпадает со значением переданного в метод.

```
my_list = [1, 3]
my_list.remove(1)
my_list # [3]
my_list.remove(4) # ValueError
```

Методы списков

list.reverse()

Перестраивает элементы списка в обратном порядке.

```
my_list = [1, 'two', 'a', 4]
my_list.reverse() # None
 my_list # [4, 'a', 'two', 1]
```

Методы списков

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

Сортирует элементы списка на месте.

key=None : Функция, принимающая аргументом элемент, используемая для получения из этого элемента значения для сравнения его с другими.

reverse=False : Флаг, указывающий следует ли производить сортировку в обратном порядке.

my_list = [1, 'two', 'a', 4, 'a'] # Попытка упорядочить/сравнить несравнимые типы вызовет исключение

my_list.sort() # TypeError: unorderable types: str() <= int() # Отсортируем «вручную», так чтобы 'a' были в конце. my_list.sort(key=**lambda** val: val == 'a') # None

Фактически мы отсортировали в соответствии

с маской [False, False, False, True, True]

my_list # ['two', 4, 1, 'a', 'a']

Методы строк

str.capitalize()

Возвращает копию строки, переводя первую буквы в верхний регистр, а остальные в нижний.

'нАЧАТЬ С ЗАГЛАВНОЙ '.capitalize() # Начать с заглавной

Методы строк

str.center(width[, fillchar])

Позиционирует по центру указанную строку, дополняя её справа и слева до указанной длины указанным символом.

width : Желаемая минимальная длина результирующей строки.

fillchar : Символ, которым следует расширять строку.
По умолчанию — пробел.

Изначальная строка не обрезается, даже если в ней меньше символов, чем указано в параметре желаемой длины.

‘ '.center(3, 'w') # www

'1'.center(2, 'w') # 1w

'1'.center(4, 'w') # w1ww

'1'.center(0, 'w') # 1

'1'.center(4) # ' 1 ‘

Символ добавляется к строке циклично сначала справа, затем слева.

Методы строк

str.count(sub[, start[, end]])

Для строки возвращает количество непересекающихся вхождений в неё указанной подстроки.

sub : Подстрока, количество вхождений которой следует вычислить.

start=0 : Позиция в строке, с которой следует начать вычислять количество вхождений подстроки.

end=None : Позиция в строке, на которой следует завершить вычислять количество вхождений подстроки.

```
my_str = 'подстрока из строк'
my_str.count('строка') # 1
 my_str.count('стр') # 2
my_str.count('стр', 0, -1) # 2
 my_str.count('стр', 8) # 1
my_str.count('стр', 1, 5) # 0
my_str.count('стр', 1, 6) # 1
```

Позиции начала и конца трактуются также как в срезах.

Методы строк

str.find(sub[, start[, end]])

Возвращает наименьший индекс, по которому обнаруживается начало указанной подстроки в исходной.

sub : Подстрока, начальный индекс размещения которой требуется определить.

start=0 : Индекс начала среза в исходной строке, в котором требуется отыскать подстроку.

end=None : Индекс конца среза в исходной строке, в котором требуется отыскать подстроку.

Если подстрока не найдена, возвращает −1.

```
my_str = 'barbarian'
my_str.find('bar') # 0
 my_str.find('bar', 1) # 3
my_str.find('bar', 1, 2) # -1
```

Методы строк

str.index(sub[, start[, end]])

Возвращает наименьший индекс, по которому обнаруживается начало указанной подстроки в исходной.

sub : Подстрока, начальный индекс размещения которой требуется определить.

start=0 : Индекс начала среза в исходной строке, в котором требуется отыскать подстроку.

end=None : Индекс конца среза в исходной строке, в котором требуется отыскать подстроку.

Работа данного метода аналогична работе str.find(), однако, если подстрока не найдена, возбуждается исключение

```
my_str = 'barbarian'
my_str.index('bar') # 0
 my_str.index('bar', 1) # 3
 my_str.index('bar', 1, 2) # ValueError
```

Методы строк

str.isalnum()

Возвращает флаг, указывающий на то, содержит ли строка только цифры и/или буквы.

Вернёт True, если в строке хотя бы один символ и все символы строки являются цифрами и/или буквами, иначе — False.

```
''.isalnum() # False
' '.isalnum() # False
 '!@#'.isalnum() # False
'abc'.isalnum() # True
'123'.isalnum() # True
'abc123'.isalnum() # True
```

Методы строк

str.isalpha()

Возвращает флаг, указывающий на то, содержит ли строка только буквы.

Вернёт True, если в строке есть хотя бы один символ, и все символы строки являются буквами, иначе — False.

```
''.isalpha() # False
' '.isalpha() # False
'!@#'.isalpha() # False
'abc'.isalpha() # True
 '123'.isalpha() # False
 'abc123'.isalpha() # False
```

Методы строк

str.isdigit()

Возвращает флаг, указывающий на то, содержит ли строка только цифры.

Вернёт True, если в строке хотя бы один символ и все символы строки являются цифрами, иначе — False.

```
''.isdigit() # False
' '.isdigit() # False
 '!@#'.isdigit() # False
 'abc'.isdigit() # False
'123'.isdigit() # True
'abc123'.isdigit() # False
```

Вложенные списки

- Для создания вложенного списка можно использовать литеральную форму записи – перечисление элементов через запятую в квадратных скобках:
- my_list = [[0], [1, 2], [3, 4, 5]]
- Иногда нужно создать вложенный список, заполненный по определенному правилу – шаблону. Например, список длиной n, содержащий списки длиной m, каждый из которых заполнен нулями.

Создание вложенных списков

Способ 1. Создадим пустой список, потом n раз добавим в него новый элемент – список длины m, составленный из нулей:

```
n, m = int(input()), int(input()) # считываем значения n и m
my_list = []
for _ in range(n):
    my_list.append([0] * m) print(my_list)
```

Если ввести значения n = 3, m = 5, то результатом работы такого кода будет:

```
[[0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0]]
```

Создание вложенных списков

Способ 2. Сначала создадим список из n элементов (для начала просто из n нулей). Затем сделаем каждый элемент списка ссылкой на другой список из m элементов, заполненный нулями:

```
n, m = int(input()), int(input()) # считываем значения n и m
my_list = [0] * n
for i in range(n):
    my_list[i] = [0] * m print(my_list)
```

Создание вложенных списков

Способ 3. Можно использовать генератор списка: создадим список из n элементов, каждый из которых будет списком, состоящих из m нулей:

```
n, m = int(input()), int(input()) # считываем значения n и m
my_list = [[0] * m
for _ in range(n)] print(my_list)
```

В этом случае каждый элемент создается независимо от остальных (заново конструируется вложенный список [0] * m для заполнения очередного элемента списка)

Считывание вложенных списков

Если элементы списка вводятся через клавиатуру (каждая строка на отдельной строке, всего n строк, числа в строке разделяются пробелами), для ввода списка можно использовать следующий код:

```
n = 4 # количество строк (элементов)
my_list = []
for _ in range(n):
    elem = [int(i) for i in input().split()] # создаем список из элементов строки
my_list.append(elem)
```

В этом примере мы используем списочный метод append(), передавая ему в качестве аргумента другой список. Так у нас получается список списков.

В результате, если на вход программе подаются строки

2 4

6 7 8 9

1 3

5 6 5 4 3 1

то в переменной my_list будет храниться список:

[[2, 4], [6, 7, 8, 9], [1, 3], [5, 6, 5, 4, 3, 1]]

Перебор и вывод элементов вложенного списка

Как мы уже знаем для доступа к элементу списка указывают индекс этого элемента в квадратных скобках. В случае двумерных вложенных списков надо указать два индекса (каждый в отдельных квадратных скобках), в случае трехмерного списка — три индекса и т. д.

Рассмотрим программный код:

```
my_list = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
print(my_list[0][0])
print(my_list[1][2])
print(my_list[2][1])
```

Результатом работы такого кода будет:

```
1
6
8
```

Когда нужно перебрать все элементы вложенного списка (например, чтобы вывести их на экран), обычно используются **вложенные циклы**.

Перебор и вывод элементов вложенного списка

```
my_list = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
for i in range(len(my_list)):
    for j in range(len(my_list[i])):
print(my_list[i][j], end=' ')
    print()
```

используем необязательный параметр end print()
перенос на новую строку Результатом работы такого кода будет:

1 2 3

4 5 6

7 8 9

Перебор и вывод элементов вложенного списка

В предыдущем примере мы перебирали **индексы элементов**, а можно сразу перебирать сами элементы вложенного списка:

```
my_list = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
for row in my_list:
    for elem in row: print(elem, end=' ')
print()
```

Результатом работы такого кода будет:

```
1 2 3
4 5 6
7 8 9
```

Обработка вложенных списков

Для обработки элементов вложенного списка, так же как и для вывода его элементов на экран как правило используются **вложенные циклы**.

Используем вложенный цикл для подсчета суммы всех чисел в списке:

```
my_list = [[1, 9, 8, 7, 4], [7, 3, 4], [2, 1]]
 total = 0
for i in range(len(my_list)):
    for j in range(len(my_list[i])):
        total += my_list[i][j]
print(total)
```

Обработка вложенных списков

Или то же самое с циклом не по индексу, а по значениям:

```
my_list = [[1, 9, 8, 7, 4], [7, 3, 4], [2, 1]]
total = 0
for row in my_list:
    for elem in row:
        total += elem
print(total)
```

Таким образом можно обработать элементы вложенного списка практически в любом языке программирования.

- Спасибо!)