Язык программирования Питон (Python)

Типы и структуры данных

Списки

Список – упорядоченный изменяемый набор объектов, в который могут одновременно входить объекты разных типов (числа, строки и другие структуры, в частности, списки и кортежи).

– задаётся перечислением его элементов в квадратных скобках через запятую

Пример: `lst = [12, 'b' ,34.6, 'derevo']`

– можно элементам списка сопоставить переменные при помощи цепочки присваиваний

(*Элемент списка и переменная будут указывать на одни и те же значения.*
Присвоение нового значения переменной никак не повлияет на элемент списка)

Пример:
```
lst=[x, s1, y, s2] = [12, 'b', 34.6, 'derevo']
lst[0]  ->  12;   x  ->  12;   x=24;   lst[0]  ->  12
```

– элементы списка нумеруются начиная с нуля;

– значения элементов списка можно изменять, добавлять элементы в список и удалять их.

Основные операции со списками

Функция или операция	Описание
`len(lst)`	Вычисляется количество элементов списка
`lst1 + lst2`	Объединение списков. Получается новый список, в котором после элементов списка `lst1` находятся элементы списка `lst2`.
`lst*n (n*lst)`	`n`-кратное повторение списка `lst.` Результат – новый список.
`lst[ i ]`	Выбор из `lst` элемента с номером i. Нумерация начинается с 0. Если i < 0, отсчёт идёт с конца (первый элемент имеет номер 0, последний имеет номер −1).
`lst[i:j:k]`	*Срез* – список, содержащий элементы списка `lst` с номерами от `i` до `j` с шагом `k` – элемент с номером `i` входит в итоговый список, а с номером `j` – не входит; – если `k` не указан (использован вариант `s[i:j]`), то элементы идут подряд
`min(lst)`	Определяет элемент с наименьшим значением в соответствии с алфавитным (словарным) порядком: сначала числа по возрастанию, затем строки, начинающиеся на цифры в порядке их возрастания, затем строки, начинающиеся на прописные буквы в алфавитном порядке, а затем строки, начинающиеся на строчные буквы также в алфавитном порядке
`max(lst)`	Определяет элемент с наибольшим значением в соответствии с алфавитным порядком

Основные операции со списками *(продолжение)*

Функция или операция	Описание

`lst[i] = x`	Замена элемента списка с номером i на значение x.
	Пример: `lst = [1, 2, 3, 'raz', 'dva']`
	`lst[2] = 'tri';  lst -> [1, 2, 'tri', 'raz', 'dva']`
	`lst[2] = [7, 8];  lst -> [1, 2, [7, 8], 'raz', 'dva']`

`del lst[i]`	Удаление из списка элемента с номером `i`. *Пример:* `lst = [1, 2, 3, 'raz', 'dva']` `del lst[2];  lst -> [1, 2, 'raz', 'dva']`

`lst[i:j] = x`	Замена среза списка `lst` на элемент или список `x`.
	строка интерпретируется как список!
	Пример: `lst = [1, 2, 3, 'raz', 'dva']`
	`lst[2:4] = 'tri';  lst -> [1, 2, 't', 'r', 'i', 'dva']`
	`lst[2:4] = 'a';  lst -> [1, 2, 'a', 'i', 'dva']`

`del lst[i:j]`	Удаление элементов, входящих в указанный срез *Пример:* `lst = [1, 2, 3, 'raz', 'dva']` `del lst[2:4];  lst -> [1, 2, 'dva']`

Основные методы списков

Метод	Описание
`lst.append(x)`	Добавление элемента `x` в конец списка `lst`. `x` не может быть списком. *Пример:* `lst=['raz','dva','tri',1,2]` `lst.append(3);  lst -> ['raz', 'dva', 'tri', 1, 2, 3]`
`lst.extend(t)`	Добавление кортежа или списка `t` в конец списка `lst` *Пример:* `lst1=[1,2,3];  lst2=['raz','dva']` `lst1.extend(lst2);  lst1 -> [1, 2, 3, 'raz', 'dva']`
`lst.count(x)`	Определение количества элементов, равных `x`, в списке `lst`. *Пример:* `lst=[1,2,3,'raz','dva','raz','dva']` `lst.count('raz');  lst -> 2`
`lst.index(x, start=0, stop=9223372036854775807)`	Определение первой слева позиции элемента `x` в списке `lst`. Если такого элемента нет, возникает сообщение об ошибке. *Пример:* `lst=[1,2,3,'raz','dva','raz','dva']` `lst.index('dva');  lst -> 4`

Основные методы списков (продолжение)

Метод	Описание
`lst.remove(x)`	Удаление элемента `x` в списке `lst` в первой слева позиции. Если такого элемента нет, возникает сообщение об ошибке. *Пример:* `lst=[1,2,3,'raz','dva','raz','dva']` `lst.remove('dva');  lst -> [1,2,3,'raz','raz','dva']`

`lst.pop(i)`	Удаление элемента с номером `i` из списка `lst`. При этом выдаётся значение этого элемента (“извлечение” элемента из списка). Если номер не указан, удаляется последний элемент.
	Пример: `lst=[1,2,3,'raz','raz','dva']`
	`lst.pop(3);  lst -> [1, 2, 3, 'raz', 'dva']`
	`lst.pop();  lst -> [1, 2, 3, 'raz']`

`lst.insert(i,x)`	Вставка элемента или списка `x` в позицию i списка `lst`. Если `i >= 0`, вставка идёт в начало списка. Если `i > len(lst)`, вставка идёт в конец списка. *Пример:* `lst=[1,2,3,'raz']` `lst.insert(3,'tri');  lst -> [1, 2, 3, 'tri', 'raz']`

Основные методы списков *(продолжение)*

Метод	Описание
`lst.sort()`	Сортировка списка по возрастанию. *Пример:* `lst=[1,2,3,15,10,6,8]` `lst.sort();  lst -> [1, 2, 3, 6, 8, 10, 15]`
`lst.reverse()`	Замена порядка следования элементов на обратный. *Пример:* `lst=[1,2,3,'tri','raz']` `lst.reverse();  lst -> ['raz', 'tri', 3, 2, 1]`
`lst.clear()`	Очищает список *Пример:* `lst=[1,2,3,'tri','raz']` `lst.clear();  lst -> []`

Встроенные функции для работы со списками:

sum() – вычисляет сумму элементов списка (кортежа)

Пример:
```
lst = [1, 2, 3, 4];   sum(lst)  ->  10
```

tuple() – преобразует строку или список в кортеж

Пример:
```
s='amamam';  t = tuple(s);
             t   ->  ('a', 'm', 'a', 'm', 'a', 'm')
```

list() – преобразование строки или кортежа в список

Пример:
```
s = 'amamam';   lst = list(s);
                lst  ->  ['a', 'm', 'a', 'm', 'a', 'm']

t = (5, 12, -3, 7);   lst = list(t);
                      lst  ->  [5, 12, -3, 7]
```

split() – делит строку по заданному символу-разделителю и создаёт список из фрагментов строки.

Пример:
```
s = 'mama myla ramu'
lst = s.split(' ')    # символ-разделитель - пробел
lst  ->  ['mama', 'myla', 'ramu']
```

join() – формирует строку из элементов списка, поставив между ними заданную строку

Пример:
```
lst = ['1', '2', '3']
s = 'nea'.join(lst);   s  ->  '1nea2nea3'
```

Замечание. *При необходимости изменения строки её преобразуют в список (**list**), изменяют, затем обратно преобразуюи в строку (**join**)*

```
delimiter = '_*_'
mylist = ['Бразилия', 'Россия', 'Индия', 'Китай']
print(delimiter.join(mylist))   -> Бразилия_*_Россия_*_Индия_*_Китай
```

sorted(L) – возвращает копию списка **L**, в котором элементы упорядочены по возрастанию. Не изменяет список **L**.

Пример:
```
e=[56.8060, 57.1578, 57.4093, 56.1843, 57.2207]
L = sorted(e)
L  ->  [56.1843, 56.806, 57.1578, 57.2207, 57.4093]
```

range(x0, x1, d) – создаёт список как числовую арифметическую прогрессию из чисел в полуоткрытом интервале **[x0, x1)** с шагом **d**

Пример:
```
list( range(0, 15, 3) )   ->  [0, 3, 6, 9, 12]
```

range(n) – создаёт список чисел от **0** до **n – 1** с шагом **1**.

range(k, n) – создаёт список чисел от **k** до **n – 1** с шагом **1**.

Проверка принадлежности величины **x** списку **range()**:

Пример:
```
x=6; x in range(0, 15, 3)        ->  True
x=6; x not in range(0, 15, 3)    ->  False
x=7; x in range(0, 15, 3)        ->  False

sum(range(10))          ->  45
sum(range(0, 15, 3))    ->  30
```

Получение последовательности чисел ***в обратном порядке***

Пример:
```
for i in range(12, 2, -2):
    print(i, end=' ')   ->  12 10 8 6 4
```

`zip()` – позволяет получить из элементов различных списков список кортежей, состоящий из соответствующих элементов списков.

Аргументами функции **`zip()`** *являются два или более списков, а результатом – список кортежей, составленных из элементов исходных списков с одинаковыми номерами (первый кортеж составляется из элементов с номерм 0, второй – из элементов с номером 1 и т.д.)*

Количество элементов в итоговом списке равно количеству элементов в самом коротком исходном списке. "Лишние" элементы других списков игнорируются.

Пример:

```
lst1=[1, 2, 3, 4];      lst2 = ['raz','dva','tri']
list(zip(lst1, lst2))  ->  [(1,'raz'), (2,'dva'), (3,'tri')]
```

map() – используется для применения одной и той же операции к элементам одного или нескольких списков или кортежей. (*Если списков (кортежей) несколько, они должны быть одинаковой длины*).

Пример:

```
def f(x):
    return x+5

lst = list(map(f,[1,3,4]))
print(lst)  ->  [6, 8, 9]
```

Пример:

```
t1 = (1 ,2 ,3);     lst1 = [5.0, 6.0 , 7.0]

lst = map(lambda x, y : x/y,  t1,  lst1)

list(lst)  ->  [0.2, 0.3333333333333333, 0.42857142857142855]
```

При использовании **map()** *чаще всего применяются так называемые* ***lambda-функции****, т.е. безымянные функции, действующие только на время конкретной операции* **map()**.

При создании ***lambda-функции*** *указывается ключевое слово* **lambda**, *затем пишутся переменные, для которых эта функция определяется и операции с этими переменными.*

После описания ***lambda-функции****, которая является первым аргументом функции* **map()** *пишутся остальные аргументы – имена списков (кортежей), с которыми надо выполнить операцию.*

Псевдонимы и копирование списков

***Замечание**. Имя объекта указывает на ту часть памяти, где хранится объект.*

Пример:

```
# Простое присваивание
shoplist = ['яблоки', 'манго', 'морковь', 'бананы']   # список покупок
mylist = shoplist                                     # указатель на тот же список

del shoplist[0]                                       # сделали первую покупку

print('shoplist:', shoplist)          ->  shoplist: ['манго', 'морковь', 'бананы']
print('mylist:', mylist)              ->  mylist: ['манго', 'морковь', 'бананы']

# Копирование при помощи полной вырезки

mylist = shoplist[:]                              # создаём копию путём полной вырезки

del shoplist[0]                                   # удаляем первый элемент списка

                                                  # теперь списки различаются
print('shoplist:', shoplist)          ->  shoplist: ['морковь', 'бананы']
print('mylist:', mylist)              ->  mylist: ['манго', 'морковь', 'бананы']
```

***Замечание**. Две переменные называются **псевдонимами**, когда они содержат одинаковые адреса памяти.*

Проверка, ссылаются ли переменные на один и тот же список

Пример:

```
x = y = [1, 2]  # создали псевдонимы
x is y        ->  True

x = [1, 2]
y = [1, 2]
x is y       ->  False
```

Вложенные списки

(Обращение к вложенному списку происходит через указание двух индексов)

Пример:

```
lst=[['A', 1], ['B',2], ['C',3]]

lst          ->  [['A', 1], ['B', 2], ['C', 3]]
lst[0]       ->  ['A', 1]
lst[0][1]    ->  1
```

К спискам применимы два вида копирования.

– ***поверхностное копирование*** *(если списки вложенные, то изменение одного влечёт изменение другого)*

```
a = [4, 3, [2, 1]]
b = a[:]
b is a        ->  False

a[0]=5        # список b не изменяется
a   ->  [5, 3, [2, 1]]
b   ->  [4, 3, [2, 1]]

a[2][0]=8     # список b изменяется
a   ->  [5, 3, [8, 1]]
b   ->  [4, 3, [8, 1]]
```

– ***глубокое копирование***

(создается новый объект и рекурсивно создаются копии всех объектов оригинала)

```
import copy

a = [ 4, 3, [2, 1] ]
b = copy.deepcopy(a)
b is a        ->  False

b[2][0]=-100      # список a не изменяется
a   ->  [4, 3, [2, 1]]
b   ->  [4, 3, [-100, 1]]
```