

Списки - операторы и функции работы с НИМИ

Мы переходим к новой теме – списки в языке Python. Что такое список и зачем он нужен? Представьте, что нам в программе нужно хранить и обрабатывать список городов, или список оценок студента, или список булевых значений, или значения функции и многое другое. Часто, когда нам нужно оперировать набором каких-либо данных, используются списки. И на этом занятии мы начнем с ними знакомиться.

Задать список в программе на Python очень просто, ставятся квадратные скобки и внутри них через запятую перечисляются его элементы:

`["Москва", "Тверь", "Вологда"]`

Города:	Казань	Тверь	Уфа	Астрахань		
Оценки:	4	3	2	2	4	5
Булевый:	True	False	False	True		
$f(x) =$	1	2	3	4	5	6

чтобы оперировать списком через переменную, используется оператор присваивания:

`marks = [2, 3, 4, 3, 5, 2]`

Давайте теперь посмотрим, как эту конструкцию этот список использовать в программировании? Например, вычислить средний балл по оценкам? Для этого нам надо уметь обращаться к отдельным элементам списка. Как это сделать? Так как список – это коллекция, то все его элементы имеют свой порядковый номер – индекс, начиная с нулевого и заканчивая последним.

индекс:	0	1	2	3	4	5
	2	3	4	3	5	2
	-6	-5	-4	-3	-2	-1

Используя эти индексы и синтаксис:

`список[индекс]`

мы можем обращаться к отдельным элементам. Например:

`marks[0]`
`marks[2]`

Причем индексирование здесь работает так же, как и со строками. Если указать несуществующий индекс:

`marks[10]`

то получим ошибку. А чтобы обратиться к последнему элементу, можно использовать отрицательный индекс:

`marks[-1]`

То есть, мы здесь также имеем наборы отрицательных индексов, идущих от конца к началу списка.

Список – изменяемый тип данных

В отличие от строк, списки в Python относятся к изменяемым типам данных. То есть, мы можем изменить ранее хранимое значение. Например, студент пересдал первую двойку на тройку и мы хотим внести это изменение в список `marks`. Сделать это можно через оператор присваивания:

```
marks[0] = 3
```

Все, теперь первое значение равно 3. Как вы помните, со строками такая операция приводила к ошибке, так как строки – это неизменяемый тип. Но со списками мы так делать можем и в этом их кардинальное отличие от строк. Список – динамическая структура данных, который может меняться в процессе работы программы.

Мало того, списку, состоящему из чисел, мы легко можем присвоить любой другой тип данных, например, строку:

```
marks[1] = "удовл."
```

Вообще списки могут содержать самые разные типы данных, например:

```
lst = ["Москва", 1320, 5.8, True,]
```

в том числе и другие, вложенные списки: `lst2 = [1, 2.5, [-1, -2, -3], 4]`

О вложенных списках мы еще будем говорить.

Если нам нужно создать пустой список, то достаточно записать квадратные скобки без элементов:

```
a = []
```

или, воспользовавшись специальной функцией: `b = list()`

которая создает новый пустой список. Если же ей в качестве аргумента указать другой список:

```
a = list([True, False])
```

то будет создан новый список с тем же самым содержимым. Также мы можем передать ей строку:

```
list("Python")
```

тогда получим список, состоящий из отдельных символов этой строки. Вообще на вход функции `list()` можно передавать любой перебираемый объект, на основе которого формируется новый список. Такие перебираемые объекты еще называются **итерируемыми**, но мы о них будем говорить на будущих занятиях.

Функции работы со списками

Язык Python содержит несколько удобных встроенных функций для работы со списками:

- `len()` – определение числа элементов в списке (длина списка);
- `max()` – для нахождения максимального значения;
- `min()` – для нахождения минимального значения;
- `sum()` – для вычисления суммы;
- `sorted()` – для сортировки коллекции.

Для начала воспользуемся уже знакомой нам функцией `len` для определения длины списка: `len(marks)`

Соответственно, для пустого списка: `len([])`

она возвращает 0.

Две из них `min()` и `max()` нам уже знакомы. Сейчас мы увидим, как их можно применять к спискам. Сформируем список значений температуры по дням города Москвы:

```
t = [23.5, 25.6, 27.3, 26.0, 30.4]
```

Теперь, чтобы найти максимальное и минимальное значения, достаточно вызвать функции: `max(t)`
`min(t)`

Для подсчета суммы всех значений, запишем функцию `sum`: `sum(t)`

А вычислить среднюю температуру можно следующим образом: `sum(t)/len(t)`

Наконец, последняя функция `sorted`, если ее вызвать с одним аргументом: `sorted(t)`

то она возвратит новый список с отсортированными значениями по неубыванию (или, как часто говорят, по возрастанию). И, обратите внимание, эта функция не меняет прежний список `t`, она именно возвращает новый список с отсортированными значениями. Очевидно, чтобы сохранить результат работы этой функции, следует использовать переменную, например, так: `t_sort = sorted(t)`

Если же нам нужно отсортировать список по невозрастанию (убыванию), то дополнительно прописывается параметр: `sorted(t, reverse=True)`

Функции `min()`, `max()` и `sorted()` работают не только с числовыми типами, но и вообще с любыми, где допустимы операторы сравнения больше и меньше. Например, создадим список из символов (строк):

```
s = list("python")
max(s)
min(s)
sorted(s)
```

И выполним все те же самые функции:

А вот функция `sum()` приведет к ошибке: `sum(s)`

Операторы списков

При работе со списками часто используются следующие операторы:

- `+` – соединение двух списков в один;
- `*` – дублирование списка;
- `in` – проверка вхождения элемента в список;
- `del` – удаление элемента списка.

Например: `[1, 2, 3] + [4, 5]` Но, вот так: `[1, 2, 3] + 4`

работать не будет, так как оператор `+` соединяет именно списки между собой, просто число или какой-либо другой тип данных записывать нельзя. В данном случае правильно будет так: `[1, 2, 3] + [4]`

Или, так: `[1, 2, 3] + [True]`

Как видите, добавлять в список можно самые разные типы данных.

Следующий оператор `*` выполняет дублирование списка указанное число раз:

```
["Я", "люблю", "Python"] * 3
```

Этот оператор работает также как и со строками, здесь можно указывать только целое число (или переменную, ссылающуюся на

целое значение). Прописывать дробные числа нельзя:

```
["Я", "люблю", "Python"] * 3.5
```

На практике можно комбинировать операторы и определять, например, такие конструкции:

```
["Я"] + ["люблю"] * 3 + ["Python"]
```

Все достаточно очевидно, гибко, наглядно и просто. Этим и знаменателен этот язык. В нем многое реализуется простыми и понятными методами, в отличие от других языков программирования.

Следующий оператор `in` позволяет определять вхождение некоторого значения в список. Делается это также просто, например:

```
lst = ["Москва", 1320, 5.8, True, 120 ]
1320 in lst
120 in lst
```

Или, можно узнать, является ли элементом списка другой список:

```
[1, 2] in lst
```

В данном случае получим `False`, но если его добавить:

```
lst = ["Москва", 1320, 5.8, True, [1, 2] ]
```

```
то [1, 2] in lst
```

вернет `True`. И так со всеми типами данных: `"Москва" in lst`

Последний оператор `del` выполняет удаление элемента списка по его индексу, например, так:

```
del lst[2]
```