Lists

Lists

```
mylist = ["apple", "banana", "cherry"]
```

Списки используются для хранения нескольких элементов в одной переменной.
Списки - один из 4 встроенных типов данных в Python, используемых для хранения коллекций данных, остальные 3 типа - это кортеж, набор и словарь, каждый из которых обладает различными свойствами и возможностями использования.
Списки создаются с помощью квадратных скобок:

```
thislist = ["apple", "banana", "cherry"]
print(thislist)
```

1. List items are ordered, changeable, and allow duplicate values. List items are indexed, the first item has index `[0]`, the second item has index `[1]` etc.
2. The list is changeable, meaning that we can change, add, and remove items in a list after it has been created.

Range of Indexes:

You can specify a range of indexes by specifying where to start and where to end the range. When specifying a range, the return value will be a new list with the specified items.

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[2:5])
```

Change List Items:

Чтобы изменить значение элементов в определенном диапазоне, определите список с новыми значениями и обратитесь к диапазону индексных номеров, в который нужно вставить новые значения:

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "mango", "juice", "pear"]
thislist[4:5] = ["blackcurrant", "watermelon"]
print(thislist)
```

Это заменил с 4 элемента по 5 элемент (то есть только "kiwi") нашими 2-мя новыми элементами "blackcurrant" и "watermelon"!

`insert() method.`

To insert a new list item, without replacing any of the existing values, we can use the `insert()` method.

The `insert()` method inserts an item at the specified index:

```
thislist = ["apple", "banana", "cherry"]
thislist.insert(2, "watermelon")
print(thislist)
```

append() method.

To add an item to the end of the list, use the append() method:

```
thislist = ["apple", "banana", "cherry"]
thislist.append("orange")
print(thislist)
```

extend() method.

To append elements from *another list* to the current list, use the `extend()` method.

```
thislist = ["apple", "banana", "cherry"]
tropical = ["mango", "pineapple", "papaya"]
thislist.extend(tropical)
print(thislist)
```

Метод extend() не обязательно должен добавлять списки, вы можете добавить любой итерируемый объект (кортежи, множества, словари и т.д.).

```
thislist = ["apple", "banana", "cherry"]
thistuple = ("kiwi", "orange")
thislist.extend(thistuple)
print(thislist)
```

remove() method.

The `remove()` method removes the specified item.

```
thislist = ["apple", "banana", "cherry"]
thislist.remove("banana")
print(thislist)
```

pop() method.

The `pop()` method removes the specified index.

```
thislist = ["apple", "banana", "cherry"]
thislist.pop(1)
print(thislist)

# If you do not specify the index, the pop() method removes the last item.
```

```
thislist = ["apple", "banana", "cherry"]
thislist.pop()
print(thislist)
```

`del() method.`

The `del` keyword also removes the specified index:

```
thislist = ["apple", "banana", "cherry"]
del thislist[0]
print(thislist)

# The del keyword can also delete the list completely.

thislist = ["apple", "banana", "cherry"]
del thislist
```

`clear() method.`

The `clear()` method empties the list. The list still remains, but it has no content.

```
thislist = ["apple", "banana", "cherry"]
thislist.clear()
print(thislist)
```

For Loop in the List

You can loop through the list items by using a `for` loop:

```
thislist = ["apple", "banana", "cherry"]
for x in thislist:
  print(x)
```

Вы также можете перебирать элементы списка, ссылаясь на их индексный номер.

Используйте функции range() и len() для создания подходящей итерационной таблицы.

```
thislist = ["apple", "banana", "cherry"]
for x in range(len(thislist)):
  print(thislist[x])
```

range() Function

Create a sequence of numbers from 0 to 5, and print each item in the sequence:

```
x = range(6)
for n in x:
  print(n)
```

While Loop in the List

Use the `len()` function to determine the length of the list, then start at 0 and loop your way through the list items by referring to their indexes.
Remember to increase the index by 1 after each iteration.

```
thislist = ["apple", "banana", "cherry"]
i = 0
while i < len(thislist):
  print(thislist[i])
  i = i + 1
```

Циклическая работа с использованием **List Comprehension**.
List Comprehension предлагает самый короткий синтаксис для выполнения циклов в списках:

```
thislist = ["apple", "banana", "cherry"]
[print(x) for x in thislist]
```

Example:

Based on a list of fruits, you want a new list, containing only the fruits with the letter "a" in the name.
Without list comprehension you will have to write a `for` statement with a conditional test inside:

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
newlist = []
for x in fruits:
  if "a" in x:
    newlist.append(x)
print(newlist)
```

With list comprehension you can do all that with only one line of code:

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
newlist = [x for x in fruits if "a" in x]
print(newlist)
[0, 1, 2, 3, 4]
```

newlist = [*expression* for *item* in *iterable* if *condition* == True]

- Iterable - The iterable can be any iterable object, like a list, tuple, set etc. You can use the `range()` function to create an iterable:
  - newlist = [x for x in range(10)]
- Same example, but with a condition:
  - newlist = [x for x in range(10) if x < 5]

Expression

Выражение является текущим элементом в итерации, но оно также является результатом, которым вы можете манипулировать до того, как он окажется как элемент списка в новом списке:

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
newlist = [x.upper() for x in fruits]
print(newlist)
['APPLE', 'BANANA', 'CHERRY', 'KIWI', 'MANGO']
```

Выражение также может содержать условия, но не как фильтр, а как способ манипулирования результатом:

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
newlist = [x if x != "cherry" else "pear" for x in fruits]
print(newlist)
['apple', 'banana', 'pear', 'kiwi', 'mango']
```

Если **x** НЕ равно **"cherry"** - то записываем этот элемент в список, а если **x** все таки равно **"cherry"** - то все равно (else) *записываем, но меняем его на "pear*"!

Sort Lists:

Объекты списка имеют метод sort(), который по умолчанию сортирует список по алфавитно-цифровому принципу, по возрастанию:

```
thislist = ["orange", "mango", "kiwi", "pineapple", "banana"]
thislist.sort()
print(thislist)
['banana', 'kiwi', 'mango', 'orange', 'pineapple']
```

To sort descending, use the keyword argument `reverse = True`:

```
thislist = ["orange", "mango", "kiwi", "pineapple", "banana"]
thislist.sort(reverse = True)
print(thislist)
['pineapple', 'orange', 'mango', 'kiwi', 'banana']
```

By default the `sort()` method is case sensitive, resulting in all capital letters being sorted before lower case letters:

```
thislist = ["banana", "Orange", "Kiwi", "cherry", "apple", "Apple"]
thislist.sort()
print(thislist)
['Apple', 'Kiwi', 'Orange', 'apple', 'banana', 'cherry']
```

К счастью, мы можем использовать встроенные функции в качестве ключевых функций при сортировке списка.
Так, если вам нужна функция сортировки без учета регистра, используйте **str.lower** в качестве ключевой функции:

```
thislist = ["banana", "Orange", "Kiwi", "cherry", "apple", "Apple"]
thislist.sort(key = str.lower)
print(thislist)
['apple', 'Apple', 'banana', 'cherry', 'Kiwi', 'Orange']
```

reverse() method.

`reverse()` method reverses the current sorting order of the elements.

```
thislist = ["banana", "Orange", "Kiwi", "cherry"]
thislist.reverse()
print(thislist)

['cherry', 'Kiwi', 'Orange', 'banana']
```

Copy a List:

Вы не можете скопировать список, просто набрав list2 = list1, потому что: list2 будет только ссылкой на list1, и изменения, сделанные в list1, автоматически будут сделаны и в list2. Существуют способы сделать копию, один из них - использовать встроенный метод List copy().

copy() method.

```
thislist = ["apple", "banana", "cherry"]
mylist = thislist.copy()
print(mylist)
```

list() method.

```
thislist = ["apple", "banana", "cherry"]
mylist = list(thislist)
print(mylist)
```

Join Lists:

There are several ways to join, or concatenate, two or more lists in Python.

1. One of the easiest ways are by using the `+` operator.

```
list1 = ["a", "b", "c"]
list2 = [1, 2, 3]
list3 = list1 + list2
print(list3)
```

2. Another way to join two lists is by appending all the items from list2 into list1, one by one:

```
list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]
for x in list2:
  list1.append(x)
print(list1)
```

3. Or you can use the `extend()` method, which purpose is to add elements from one list to another list:

```
list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]
list1.extend(list2)
print(list1)
```

List Methods:

Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list