Python → Collections → Operations with list

Theory: Operations with list

18 minutes 7 / 12 problems solved

Start practicing

6969 users solved this topic. Latest completion was 6 minutes ago.

You already know how to create lists (even empty ones), so, no wonder, that you may want to change your lists somehow. There are lots of things to do with a list, you can read about them in the Python Data Structures documentation. In this topic, we will discuss only the basic functions.

§1. Adding one element

A list is a **dynamic collection**, it means you can add and remove elements. To take a closer look, let's create an empty list of dragons.

```
dragons = []  # we do not have dragons yet
```

What is next? The first thing that comes to mind is, of course, **to add new elements** to the list.

To add a new element to the end of an existing list, you need to invoke the `list.append(element)` function. It takes only a single argument, so this way you can add only one element to the list at a time.

```
dragons.append('Rudror')
dragons.append('Targiss')
dragons.append('Coporth')
```

Now you have three dragons, and they are ordered the way you added them:

```
print(dragons)  # ['Rudror', 'Targiss', 'Coporth']
```

§2. Adding several elements

There is the `list.extend(another_list)` operation that adds all the elements from another iterable to the end of a list.

```
numbers = [1, 2, 3, 4, 5]
numbers.extend([10, 20, 30])
print(numbers)  # [1, 2, 3, 4, 5, 10, 20, 30]
```

Be careful — if you use `list.append(another_list)` instead of `list.extend(another_list)`, it adds the entire list as an element:

```
numbers = [1, 2, 3, 4, 5]
numbers.append([10, 20, 30])
print(numbers)  # [1, 2, 3, 4, 5, [10, 20, 30]]
```

Alternatively, to merge two lists, you can just add one to another:

```
numbers_to_four = [0, 1, 2, 3, 4]
numbers_from_five = [5, 6, 7, 8, 9]
numbers = numbers_to_four + numbers_from_five
print(numbers)  # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

If you need a list with repeating elements, you can create a list with the repeating pattern, and then just multiply it by any number. This is particularly useful when you want to create a list of a specific length with the same value:

```
pattern = ['a', 'b', 'c']
patterns = pattern * 3
print(patterns)  # ['a', 'b', 'c', 'a', 'b', 'c', 'a', 'b', 'c']

one = [1]
ones = one * 7
print(ones)  # [1, 1, 1, 1, 1, 1, 1]
```

Current topic:

✓ Operations with list Stage 3 6★

Topic depends on:

✓ Boolean logic Stage 1 16★

✓ List Stage 1 13★

✓ Indexes Stage 3 7★

✓ Arguments Stage 1 7★

Topic is required for:

Conversion to boolean

✓ Default arguments

✓ Stack in Python

Aggregations and Ordering

§3. Removing elements

The opposite of adding elements — **deleting them** — can be done in three ways. Let's have a look at them.

First, we can use the `list.remove(element)` operation.

```
dragons.remove('Targiss')
print(dragons)  # ['Rudror', 'Coporth']
```

If the element we want to delete occurs several times in the list, only the first instance of that element is removed.

```
dragons = ['Rudror', 'Targiss', 'Coporth', 'Targiss']
dragons.remove('Targiss')
print(dragons)  # ['Rudror', 'Coporth', 'Targiss']
```

The other two ways remove elements by their **indexes** rather than the values themselves. The `del` keyword deletes any kind of objects in Python, so it can be used to remove specific elements in a list:

```
dragons = ['Rudror', 'Targiss', 'Coporth']
del dragons[1]
print(dragons)  # ['Rudror', 'Coporth']
```

Finally, there is the `list.pop()` function. If used without arguments, it removes and returns **the last element** in the list.

```
dragons = ['Rudror', 'Targiss', 'Coporth']
last_dragon = dragons.pop()
print(last_dragon)  # 'Coporth'
print(dragons)      # ['Rudror', 'Targiss']
```

Alternatively, we can specify the index of the element we want to remove and return:

```
dragons = ['Rudror', 'Targiss', 'Coporth']
first_dragon = dragons.pop(0)
print(first_dragon)  # 'Rudror'
print(dragons)       # ['Targiss', 'Coporth']
```

§4. Inserting elements at a specified position

At the beginning of this topic, we have learned how to add new elements to the end of a list. If we want to add a new element in the middle, we use the `list.insert(position, element)` operation. The first argument is the index of the element **before** which the new element is going to be inserted; so `list.insert(0, element)` inserts an element to the beginning of the list, and `list.insert(len(list), element)` is completely similar to `list.append(element)`.

Here is an example:

```
years = [2016, 2018, 2019]
years.insert(1, 2017)           # [2016, 2017, 2018, 2019]
years.insert(0, 2015)           # [2015, 2016, 2017, 2018, 2019]
years.insert(len(years), 2020)  # [2015, 2016, 2017, 2018, 2019, 2020]
```

Now, you can fill any empty list with something useful!

§5. Membership testing in a list

Another thing that can be quite useful is checking if an item is present in the list. It can be done simply by using `in` and `not in` operators:

```
catalog = ['yogurt', 'apples', 'oranges', 'bananas', 'milk', 'cheese']

print('bananas' in catalog)      # True

product = 'lemon'
print(product in catalog)        # False
print(product not in catalog)    # True
```

§6. Searching specific elements

Sometimes, knowing that the specified element is in the list is not enough; we may want to get more information about it — how many times the element occurs in the list and at which position.

The method `count()` can help with the quantity:

```
grades = [10, 5, 7, 9, 5, 10, 9]
print(grades.count(5))  # 2
```

We can use the method `index()` to get the position of the element. It finds the index of the **first occurrence** of the element in the list:

```
print(grades.index(7))   # 2
print(grades.index(10))  # 0
```

We can also specify the interval for searching: `list.index(element, start, end)`.

```
print(grades.index(9, 2, 5))  # 3


# if we don't specify the end of the interval, it automatically equals the end of the list
print(grades.index(10, 1))    # 5
```

Be careful — the `end` index is not included in the interval.

It is also good to know that if the element we are looking for is not in the list, the method will cause an **error**:

```
print(grades.index(8))  # ValueError: 8 is not in list
```

Our discussion of the basic operations with lists has come to its end. If you need more information, check out the [Python Data Structures documentation](https://docs.python.org).

Report a typo

571 users liked this theory. **1** didn't like it. **What about you?**

Start practicing

Table of contents:

Comments (16) Hints (0) Useful links (0) Show discussion