Theory: List

10 minutes 12 / 12 problems solved

Start practicing

22467 users solved this topic. Latest completion was **about 1 hour ago**.

In your programs, you often need to group several elements in order to process them as a single object. For this, you will need to use different collections. One of the most useful collections in Python is a **list**. It is one of the most important things in Python.

§1. Creating and printing lists

Look at a simple list that stores several names of dogs' breeds:

```
dog_breeds = ['corgi', 'labrador', 'poodle', 'jack russell']
print(dog_breeds)  # ['corgi', 'labrador', 'poodle', 'jack russell']
```

In the first line, we use square brackets to create a list that contains four elements and then assign it to the `dog_breeds` variable. In the second line, the list is printed through the variable's name. All the elements are printed in the same order as they were stored in the list because lists are **ordered**.

Here is another list that contains five integers:

```
numbers = [1, 2, 3, 4, 5]
print(numbers)  # [1, 2, 3, 4, 5]
```

Another way to create a list is to invoke the `list` function. It is used to create a list out of an **iterable** object: that is, a kind of object where you can get its elements one by one. The concept of iterability will be explained in detail further on, but let's look at the examples below:

```
list_out_of_string = list('danger!')
print(list_out_of_string)  # ['d', 'a', 'n', 'g', 'e', 'r', '!']

list_out_of_integer = list(235)  # TypeError: 'int' object is not iterable
```

So, the `list` function creates a list containing each element from the given iterable object. For now, remember that a **string** is an example of an **iterable** object, and an **integer** is an example of a **non-iterable** object. A **list** itself is also an **iterable** object.

Let's also note the difference between the `list` function and creating a list using square brackets:

```
multi_element_list = list('danger!')
print(multi_element_list)  # ['d', 'a', 'n', 'g', 'e', 'r', '!']

single_element_list = ['danger!']
print(single_element_list)  # ['danger!']
```

The square brackets and the `list` function can also be used to create **empty lists** that do not have elements at all.

```
empty_list_1 = list()
empty_list_2 = []
```

In the following topics, we will consider how to fill empty lists.

§2. Features of lists

Lists can store **duplicate values** as many times as needed.

```
on_off_list = ['on', 'off', 'on', 'off', 'on']
print(on_off_list)  # ['on', 'off', 'on', 'off', 'on']
```

Current topic:

- ✓ List 13 Stage 1

Topic depends on:

- ✓ Variables 17 Stage 1

Topic is required for:

- ✓ Indexes 7 Stage 3
- ✓ Set
- ✓ Operations with list 6 Stage 3
- ✓ For loop 12 Stage 1
- ✓ Random module 4 Stage 4
- ✓ Class 3 Stage 1
- ✓ Sorting a list
- Series
- Launching web server Stage 1
- Intro to NumPy
- Data types in NumPy

Table of contents:

Another important thing about lists is that they can contain **different types** of elements. So there are neither restrictions, nor fixed list types, and you can add to your list any data you want, like in the following example:

```
different_objects = ['a', 1, 'b', 2]
```

§3. Length of a list

Sometimes you need to know how many elements are there in a list. There is a built-in function called `len` that can be applied to any **iterable** object, and it returns simply the **length** of that object

So, when applied to a list, it returns the number of elements in that list.

```
numbers = [1, 2, 3, 4, 5]
print(len(numbers))  # 5

empty_list = list()
print(len(empty_list))  # 0

single_element_list = ['danger!']
print(len(single_element_list))  # 1

multi_elements_list = list('danger!')
print(len(multi_elements_list))  # 7
```

In the example above, you can see how the `len()` function works. Again, pay attention to the difference between `list()` and `[]` as applied to strings: it may not result in what you expected.

§4. Recap

As a recap, we note that lists are:

- **ordered**, i.e. each element has a fixed position in a list;
- **iterable**, i.e. you can get their elements one by one;
- able to store **duplicate values**;
- able to store **different types of elements**.

Report a typo

2085 users liked this theory. 35 didn't like it. **What about you?**

Start practicing

Comments (28)   Hints (1)   Useful links (0)   Show discussion