Theory: Nested lists

34 minutes 12 / 13 problems solved

Start practicing

5545 users solved this topic. Latest completion was about 4 hours ago.

A list in Python may contain any objects as its elements, including other lists – they are called **nested lists**. Here is a couple of examples of nested lists:

```
nested_letters  = ['a', 'b', ['c', 'd'], 'e']
nested_numbers = [[1], [2], [3]]
```

§1. Accessing elements of nesting lists

Like with regular lists, elements of a nested list can be accessed by indexes. Note that the nested list still counts as a single element in its parent list.

```
numbers = [1, [2, 3], 4]
nested_numbers = numbers[1]

print(nested_numbers)     # [2, 3]
print(nested_numbers[1])  # 3
```

In this example, we obtain the second element of `numbers` by index 1. This element is the nested list `[2, 3]`. Then we print the second element of `nested_numbers` by index 1. This element is 3.

It is also possible to access an element of a nested list without an additional variable using a sequence of square brackets.

```
lists = [0, [1, [2, 3]]]
print(lists[1][1][0])   # 2
```

Basically, we go deeper from the outer list to the innermost when indexing. Naturally, if we ask for an element at the level that doesn't exist, we'll get an error:

```
print(lists[1][1][0][1])  # TypeError: 'int' object is not subscriptable
```

Just as if we were accessing an element that doesn't exist, at the level that does exist:

```
print(lists[3])  # IndexError: list index out of range
```

§2. Matrices

Nested lists are a convenient way to represent a matrix. For example, the matrix

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

might be represented as:

```
M = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

Note that in such a list lengths of all nested lists must be the same, and they are equal to the dimension of the matrix.

When we want to extract an element from the matrix, e.g. element *M[1][2] = 6*, the first index selects the row, and the second index selects the column. However, as you know, it differs from mathematical representation in that numbering here starts from zero, rather than from one.

§3. Nested list comprehension

Current topic:

✓ Nested lists

Topic depends on:

✓ Indexes Stage 3 7★

✓ List comprehension 5★

Topic is required for:

Copy of an object

XML in Python

To iterate over nested lists, we can use **nested list comprehensions**. It is basically a combination of two or more list comprehensions and it is quite similar to nested "for" loops. To illustrate basic syntax for this kind of comprehension, let's consider an example.

Imagine a school as a list of classes that are, in turn, lists of students (their names).

```
# original list
school = [["Mary", "Jack", "Tiffany"],
          ["Brad", "Claire"],
          ["Molly", "Andy", "Carla"]]
```

If you want to create a list of all students in all classes without the list comprehension it would look like this:

```
student_list = []
for class_group in school:
    for student in class_group:
        student_list.append(student)
```

Alternatively, we can also use a comprehension with a double `for` loop, then it would look like this:

```
student_list = [student for class_group in school for student in class_group]
```

In both cases the result is going to be the same:

```
print(student_list)
# result: ["Mary", "Jack", "Tiffany", "Brad", "Claire", "Molly", "Andy", "Carla"]
```

In this case, the order of the `for` loops is the same as in the notation with indentation: first the outer loop, and then the inner loop.

However, such a method may be less clear than the one without list comprehension, especially in cases when we need to use more than two `for` loops: it can make the code unreadable and counter-intuitive.

Consider the following line of code:

```
matrix = [[j for j in range(5)] for i in range(2)]
```

It's not that easy to understand what the created matrix will look like. Compare to when we will put it this way:

```
matrix = []

for i in range(2):

    # create empty row (a sublist inside our list)
    matrix.append([])

    for j in range(5):
        matrix[i].append(j)
```

It is much more readable, and now it is clear that the matrix will look like:

```
matrix = [[0, 1, 2, 3, 4],
          [0, 1, 2, 3, 4]]
```

It's important to bear in mind that shorter code does not imply better one, so you shouldn't misuse list comprehension when working with nested lists.

Lists as such are a very useful type of container in Data Structures, and now you know how to store multiple inner lists inside an outer one, how to reach them and work with them.

Report a typo

Table of contents:

486 users liked this theory. 16 didn't like it. What about you?

😍 🙂 😐 🙁 😡

Start practicing

Comments (9) Hints (0) Useful links (0) Show discussion