note.nkmk.me

Unpack a tuple / list in Python

Posted: 2019-06-24 / Tags: Python, List

In Python, elements of tuples and lists can be assigned to multiple variables. It is called sequence unpacking.

- 5. Data Structures — Python 3.7.4rc1 documentation

This article describes the following contents.

- Basics of unpacking a tuple and a list
- Unpack a nested tuple and list
- Unpack using `_` (underscore)
- Unpack using `*` (asterisk)

Please refer to the following article for the case of expanding tuples, lists, and dictionaries (`dict`) as arguments with `*` (asterisk).

- **Related:** Expand and pass list, tuple, dict to function arguments in Python

Basics of unpacking a tuple and a list

If you write variables on the left side separated by commas `,`, elements of a tuple and a list on the right side will be assigned to each variable. The same applies to tuples and lists (the following examples are described by tuples).

```
t = (0, 1, 2)

a, b, c = t

print(a)
print(b)
print(c)
# 0
# 1
# 2
```

```
l = [0, 1, 2]

a, b, c = l

print(a)
print(b)
print(c)
# 0
# 1
# 2
```

source: tuple_list_unpack.py

Because parentheses of tuples can be omitted, multiple values can be assigned to multiple variables in one line as follows.

- **Related:** Multiple assignment in Python: Assign multiple values or the same value to multiple variables

```
a, b = 0, 1

print(a)
print(b)
# 0
# 1
```

source: tuple_list_unpack.py

An error occurs if the number of variables does not match the number of elements.

```
# a, b = t
# ValueError: too many values to unpack (expected 2)

# a, b, c, d = t
# ValueError: not enough values to unpack (expected 4, got 3)
```

source: tuple_list_unpack.py

If the number of variables is less than the number of elements, it is possible to add an asterisk `*` to the variable name and assign the remaining elements as a list. This will be described later.

Unpack a nested tuple and list

You can also unpack a nested tuple and list. If you want to expand the inner element, enclose the variable with `()` or `[]`.

```
t = (0, 1, (2, 3, 4))

a, b, c = t

print(a)
print(b)
print(c)
# 0
# 1
# (2, 3, 4)

print(type(c))
# <class 'tuple'>
```

```
a, b, (c, d, e) = t

print(a)
print(b)
print(c)
print(d)
print(e)
# 0
# 1
# 2
# 3
# 4
```

source: tuple_list_unpack.py

Unpack using _ (underscore)

By convention, unnecessary values may be assigned to underscores _ in Python. It does not have a grammatical special meaning, but is simply assigned to a variable named _.

```
t = (0, 1, 2)

a, b, _ = t

print(a)
print(b)
print(_)
# 0
# 1
# 2
```

source: tuple_list_unpack.py

Unpack using `*` (asterisk)

If the number of variables is less than the number of elements, adding an asterisk `*` to the variable name will assign the elements together as a list.

It is implemented in Python 3 and can not be used in Python 2.

The elements from the beginning and the end are assigned to variables without `*`, and the remaining elements are assigned as a list to variables with `*`.

```
t = (0, 1, 2, 3, 4)

a, b, *c = t

print(a)
```

```
print(b)
print(c)
# 0
# 1
# [2, 3, 4]

print(type(c))
# <class 'list'>

a, *b, c = t

print(a)
print(b)
print(c)
# 0
# [1, 2, 3]
# 4

*a, b, c = t

print(a)
print(b)
print(c)
# [0, 1, 2]
# 3
# 4
```

source: tuple_list_unpack.py

For example, when it is desired to assign only the first two elements of a tuple or a list to variables, the underscore `_` may be used for unnecessary parts.

```
a, b, *_ = t

print(a)
```

```
print(b)
print(_)
# 0
# 1
# [2, 3, 4]
```

source: tuple_list_unpack.py

The same process can be written as:

```
a, b = t[0], t[1]

print(a)
print(b)
# 0
# 1
```

source: tuple_list_unpack.py

You can add `*` to only one variable.

If there are multiple variables with `*`, it can not be determined how many elements are to be assigned, so `SyntaxError` occurs.。

```
# *a, b, *c = t
# SyntaxError: two starred expressions in assignment
```

source: tuple_list_unpack.py

Note that even if there is only one element assigned to a variable with `*`, it is assigned as a list.

```
t = (0, 1, 2)

a, b, *c = t

print(a)
print(b)
print(c)
# 0
# 1
# [2]

print(type(c))
# <class 'list'>
```

source: tuple_list_unpack.py

If there are no extra elements, an empty list is assigned.

```
a, b, c, *d = t

print(a)
print(b)
print(c)
print(d)
# 0
# 1
# 2
# []
```

source: tuple_list_unpack.py

Related Categories

- Python
- List

Related Articles

- How to slice a list, string, tuple in Python
- Shuffle a list, string, tuple in Python (random.shuffle, sample)

- Cartesian product of lists in Python (itertools.product)
- How to flatten a list of lists in Python
- List comprehensions in Python
- zip() in Python: Get elements from multiple lists
- Transpose 2D list in Python (swap rows and columns)
- Convert lists and tuples to each other in Python
- Convert pandas.DataFrame, Series and list to each other
- Sort a list, string, tuple in Python (sort, sorted)
- Remove an item from a list in Python (clear, pop, remove, del)
- Expand and pass list, tuple, dict to function arguments in Python
- Add an item to a list in Python (append, extend, insert)
- enumerate() in Python: Get the element and index from a list
- in operator in Python (for list, string, dictionary, etc.)
