Python Unpacking Tuple

Summary: in this tutorial, you'll learn how to unpack tuples in Python.

Reviewing Python tuples

Python defines a tuple (https://www.pythontutorial.net/python-basics/python-tuples/) using commas (`,`), not parentheses `()` . For example, the following defines a tuple with two elements:

```
1,2
```

Python uses the parentheses to make the tuple clearer:

```
(1, 2)
```

Python also uses the parentheses to create an empty tuple:

```
()
```

In addition, you can use the `tuple()` constructor like this:

```
tuple()
```

To define a tuple with only one element, you still need to use a comma. The following example illustrates how to define a tuple with one element:

```
1,
```

It's equivalent to the following:

```
(1, )
```

Note that the following is an integer (https://www.pythontutorial.net/advanced-python/python-integers/) , not a tuple:

```
(1)
```

Unpacking a tuple

Unpacking a tuple means splitting the tuple's elements into individual variables (https://www.pythontutorial.net/python-basics/python-variables/) . For example:

```
x, y = (1, 2)
```

The left side:

```
x, y
```

is a tuple of two variables `x` and `y` .

The right side is also a tuple of two integers `1` and `2` .

The expression assigns the tuple elements on the right side (1, 2) to each variable on the left side (x, y) based on the relative position of each element.

In the above example, `x` will take `1` and `y` will take `2`.

See another example:

```
x, y ,z = 10, 20, 30
```

The right side is a tuple of three integers `10`, `20`, and `30`. You can quickly check its type as follows:

```
numbers = 10, 20, 30
print(type(numbers))
```

Output:

```
<class 'tuple'>
```

In the above example, the `x`, `y`, and `z` variables will take the values `10`, `20`, and `30` respectively.

Using unpacking tuple to swap values of two variables

Traditionally, to swap the values of two variables, you would use a temporary variable like this:

```
x = 10
y = 20

print(f'x={x}, y={y}')

tmp = x
x = y
y = tmp
```

```
print(f'x={x}, y={y}')
```

Output:

```
x=10, y=20
x=20, y=10
```

In Python, you can use the unpacking tuple syntax to achieve the same result:

```
x = 10
y = 20

print(f'x={x}, y={y}')

x, y = y, x

print(f'x={x}, y={y}')
```

Output:

```
x=10, y=20
x=20, y=10
```

The following expression swaps the values of two variables, x and y.

```
x, y = y, x
```

In this expression, Python evaluates the right-hand side first and then assigns the variable from the left-hand side to the values from the right-hand side.

ValueError: too many values to unpack

The following example unpacks the elements of a tuple into variables. However, it'll result in an error:

```
x, y = 10, 20, 30
```

Error:

```
ValueError: too many values to unpack (expected 2)
```

This error is because the right-hand side returns three values while the left-hand side only has two variables.

To fix this, you can add a `_` variable:

```
x, y, _ = 10, 20, 30
```

The `_` variable is a regular variable in Python. By convention, it's called a dummy variable.

Typically, you use the dummy variable to unpack when you don't care and use its value afterward.

Extended unpacking using the * operator

Sometimes, you don't want to unpack every single item in a tuple. For example, you may want to unpack the first and second elements. In this case, you can use the `*` operator. For example:

```
r, g, *other = (192, 210, 100, 0.5)
```

Output:

```
192
210
[100, 0.5]
```

In this example, Python assigns `192` to `r`, `210` to `g`. Also, Python packs the remaining elements `100` and `0.5` into a list and assigns it to the `other` variable.

Notice that you can only use the `*` operator once on the left-hand side of an unpacking assignment.

The following example results in error:

```
x, y, *z, *t = (10, 20, 30, '10:30')
```

Error:

```
SyntaxError: two starred expressions in assignment
```

Using the * operator on the right hand side

Python allows you to use the `*` operator on the right-hand side. Suppose that you have two tuples:

```
odd_numbers = (1, 3, 5)
even_numbers = (2, 4, 6)
```

The following example uses the `*` operator to unpack those tuples and merge them into a single tuple:

```
numbers = (*odd_numbers, *even_numbers)
print(numbers)
```

Output:

```
(1, 3, 5, 2, 4, 6)
```

Summary

- Python uses the commas (`,`) to define a tuple, not parentheses.
- Unpacking tuples means assigning individual elements of a tuple to multiple variables.

- Use the `*` operator to assign remaining elements of an unpacking assignment into a list and assign it to a variable.