# Extended Unpacking

## Using the * and ** Operators

## The use case for *

Much of this section applies to Python >= 3.5

We don't always want to unpack every single item in an iterable

We may, for example, want to unpack the first value, and then unpack the remaining values into another variable

```
l = [1, 2, 3, 4, 5, 6]
```

We can achieve this using slicing:

```
a = l[0]
b = l[1:]
```

or, using simple unpacking:

```
a, b = l[0], l[1:]
```

(aka parallel assignment)

We can also use the * operator:

```
a, *b = l
```

Apart from cleaner syntax, it also works with any iterable, not just sequence types!

## Usage with ordered types

```
a, *b = [-10, 5, 2, 100]
```
a = -10  b = [5, 2, 100]

```
a, *b = (-10, 5, 2, 100)
```
a = -10  b = [5, 2, 100]

this is still a list!

this is also a list!

```
a, *b = 'XYZ'
```
a = 'X'  b = ['Y', 'Z']

The following also works:

```
a, b, *c = 1, 2, 3, 4, 5
```
a = 1  b = 2  c = [3, 4, 5]

```
a, b, *c, d = [1, 2, 3, 4, 5]
```
a = 1  b = 2  c = [3, 4]  d = 5

```
a, *b, c, d = 'python'
```
a = 'p'  b = ['y', 't', 'h']
c = 'o'  d = 'n'

The * operator can only be used once in the LHS an unpacking assignment

For obvious reason, you cannot write something like this:

```
a, *b, *c = [1, 2, 3, 4, 5, 6]
```

Since both *b and *c mean "the rest", both cannot exhaust the remaining elements

## Usage with ordered types

We have seen how to use the * operator in the LHS of an assignment to unpack the RHS

```
a, *b, c = {1, 2, 3, 4, 5}
```

However, we can also use it this way:

```
l1 = [1, 2, 3]
l2 = [4, 5, 6]
l = [*l1, *l2]     → l = [1, 2, 3, 4, 5, 6]
```

```
l1 = [1, 2, 3]
l2 = 'XYZ'
l = [*l1, *l2]     → l = [1, 2, 3, 'X', 'Y', 'Z']
```

## Usage with **un**ordered types

Types such as sets and dictionaries have no ordering

```
s = {10, -99, 3, 'd'}
print(s)                    → {10, 3, 'd', -99}
```

Sets and dictionary keys are still iterable, but iterating has no guarantee of preserving the order in which the elements were created/added

But, the * operator still works, since it works with any iterable

```
s = {10, -99, 3, 'd'}
a, *b, c = s          a = 10    b = [3, 'd']    c = -99
```

In practice, we rarely unpack sets and dictionaries directly in this way.

## Usage with **un**ordered types

It is useful though in a situation where you might want to create single collection containing all the items of multiple sets, or all the keys of multiple dictionaries

```
d1 = {'p': 1, 'y': 2}

d2 = {'t': 3, 'h': 4}

d3 = {'h': 5, 'o': 6, 'n': 7}
```

Note that the key `'h'` is in both `d2` and `d3`

```
l = [*d1, *d2, *d3]   → ['p', 'y', 't', 'h', 'h', 'o', 'n']

s = {*d1, *d3, *d3}   → {'p', 'y', 't', 'h, 'o', 'n'}
```

(order is not guaranteed)

## The ** unpacking operator

When working with dictionaries we saw that * essentially iterated the keys

```
d = {'p': 1, 'y': 2, 't': 3, 'h': 4}
a, *b = d
```

```
a = 'p'    b = ['y', 't', 'h']
```

(again, order is not guaranteed)

We might ask the question: can we unpack the key-value pairs of the dictionary?

Yes!

We need to use the ** operator

Using **

```
d1 = {'p': 1, 'y': 2}

d2 = {'t': 3, 'h': 4}

d3 = {'h': 5, 'o': 6, 'n': 7}
```

Note that the key `'h'` is in both `d2` and `d3`

```
d = {**d1, **d2, **d3}
```

(note that the ** operator cannot be used in the LHS of an assignment)

→ `{'p': 1, 'y': 2, 't': 3, 'h': 5, 'o': 6, 'n': 7}`

Note that the value of `'h'` in `d3` "overwrote" the first value of `'h'` found in `d2`

(order not guaranteed)

## Using **

You can even use it to add key-value pairs from one (or more) dictionary into a dictionary literal:

```
d1 = {'a': 1, 'b': 2}

{'a': 10, 'c': 3, **d1}    → {'a': 1, 'b': 2, 'c': 3}

{**d1, 'a': 10, 'c': 3}    → {'a': 10, 'b': 2, 'c': 3}
```

(order not guaranteed)

## Nested Unpacking

Python will support nested unpacking as well.

```
l = [1, 2, [3, 4]]
```

Here, the third element of the list is itself a list.

We can certainly unpack it this way:

```
a, b, c = l
```

```
a = 1   b = 2   c = [3, 4]
```

We could then unpack `c` into `d` and `e` as follows:

```
d, e = c
```

```
d = 3   e = 4
```

Or, we could simply do it this way:

```
a, b, (c, d) = [1, 2, [3, 4]]
```

```
a = 1   b = 2
c = 3   d = 4
```

Since strings are iterables too:

```
a, *b, (c, d, e) = [1, 2, 3, 'XYZ']
```

```
a = 1   b = [2, 3]
```

```
c, d, e = 'XYZ'
```

→ `c = 'X'   d = 'Y'   e = 'Z'`

The * operator can only be used once in the LHS an unpacking assignment

How about something like this then?

```
a, *b, (c, *d) = [1, 2, 3, 'python']
```

Although this looks like we are using * twice in the same expression, the second * is actually in a nested unpacking – so that's OK

```
a = 1    b = [2, 3]    c, *d = 'python'
```

→ `c = 'p'`

`d = ['y', 't', 'h', 'o', 'n']`

Try doing the same thing using slicing…

# Code