UNPACKING ITERABLES

A Side Note on Tuples

`(1, 2, 3)`

What defines a tuple in Python, is not ( ), but ,

1, 2, 3 is also a tuple → (1, 2, 3)

The ( ) are used to make the tuple clearer

To create a tuple with a single element:

`(1)` will not work as intended → `int`

`1,` or `(1, )` → `tuple`

The only exception is when creating an empty tuple: `( )` or `tuple( )`

Packed Values

Packed values refers to values that are bundled together in some way

Tuples and Lists are obvious

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

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

Even a string is considered to be a packed value:

```
s = 'python'
```

Sets and dictionaries are also packed values:

```
set1 = {1, 2, 3}

d = {'a': 1, 'b': 2, 'c': 3}
```

In fact, any iterable can be considered a packed value

Unpacking Packed Values

Unpacking is the act of splitting packed values into individual variables contained in a list or tuple

`a, b, c = [1, 2, 3]`   3 elements in `[1, 2, 3]`   → need 3 variables to unpack

this is actually a tuple of 3 variables: `a`, `b` and `c`

```
a → 1      b → 2      c → 3
```

The unpacking into individual variables is based on the relative positions of each element

Does this remind you of how positional arguments were assigned to parameters in function calls?

Unpacking other Iterables

```
a, b, c = 10, 20, 'hello'          → a = 10    b = 20    c = 'hello'
```

this is actually a tuple containing 3 values

```
a, b, c = 'XYZ'                    → a = 'X'    b = 'Y'    c = 'Z'
```

instead of writing

```
a = 10
b = 20
```

we can write

```
a, b = 10, 20
```

In fact, unpacking works with any iterable type

```
for e in 10, 20, 'hello'           → loop returns 10, 20, 'hello'

for e in 'XYZ'                     → loop returns 'X', 'Y', 'Z'
```

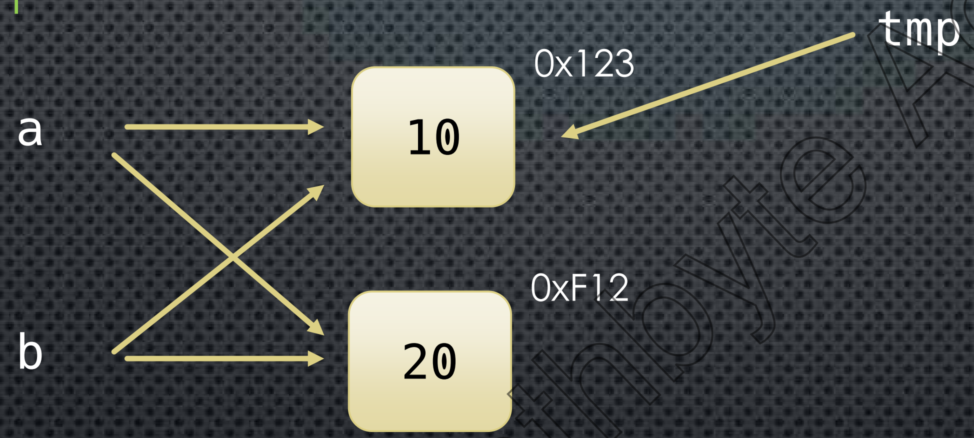
Simple Application of Unpacking

swapping values of two variables

```
a = 10        →        b = 20
b = 20                 a = 10
```

"traditional" approach

```
tmp = a
a = b
b = tmp
```



using unpacking

```
a, b = b, a
```

this works because in Python, the entire RHS is evaluated first and completely

then assignments are made to the LHS

Unpacking Sets and Dictionaries

```
d = {'key1': 1, 'key2': 2, 'key3': 3}
```

`for e in d` → e iterates through the keys: `'key1', 'key2', 'key3'`

so, when unpacking d, we are actually unpacking the keys of d

```
a, b, c = d          → a = 'key1', b = 'key2', c='key3'
                or   → a = 'key2', b = 'key1', c='key3'
                or   → a = 'key3', b = 'key1', c='key2'
                     etc…
```

Dictionaries (and Sets) are unordered types.

They can be iterated, but there is no guarantee the order of the results will match your literal!

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

Example using Sets

```
s = {'p', 'y', 't', 'h', 'o', 'n'}
```

```
for c in s:
    print(c)
```

$\rightarrow$

```
p
t
h
n
o
y
```

```
a, b, c, d, e, f = s
```

```
a = 'p'

b = 't'

c = 'h'

   …

f = 'y'
```

Code