GROUPING

Grouping

Sometimes we want to loop over an iterable of elements

but we want to group those elements as we iterate through them

Suppose we have an iterable containing tuples, and we want to group based on the first element of each tuple

```
(1, 10, 100) ┐
(1, 11, 101) ├ group 1
(1, 12, 102) ┘
(2, 20, 200) ┐
(2, 21, 201) ┘ group 2
(3, 30, 300) ┐
(3, 31, 301) ├ group 3
(3, 32, 302) ┘
```

We would like to iterate using this kind of approach:

```
for key, group in groups:
    print(key)
    for item in group:
        print(item)
```

```
key → 1
(1, 10, 100)
(1, 11, 101)
(1, 12, 102)

key → 2
(2, 20, 200)
(2, 21, 201)

key → 3
(3, 30, 300)
(3, 31, 301)
(3, 32, 302)
```

`itertools.groupby(data, [keyfunc])` → lazy iterator

The `groupby` function allows us to do precisely that

→ normally specify `keyfunc` which calculates the key we want to use for grouping

```
iterable
(1, 10, 100)
(1, 11, 101)
(1, 12, 102)
(2, 20, 200)
(2, 21, 201)
(3, 30, 300)
(3, 31, 301)
(3, 32, 302)
```

Here we want to group based on the 1st element of each tuple

→ grouping key `lambda x: x[0]`

`groupby(iterable, lambda x: x[0])`

→ iterator → of tuples `(key, sub_iterator)`

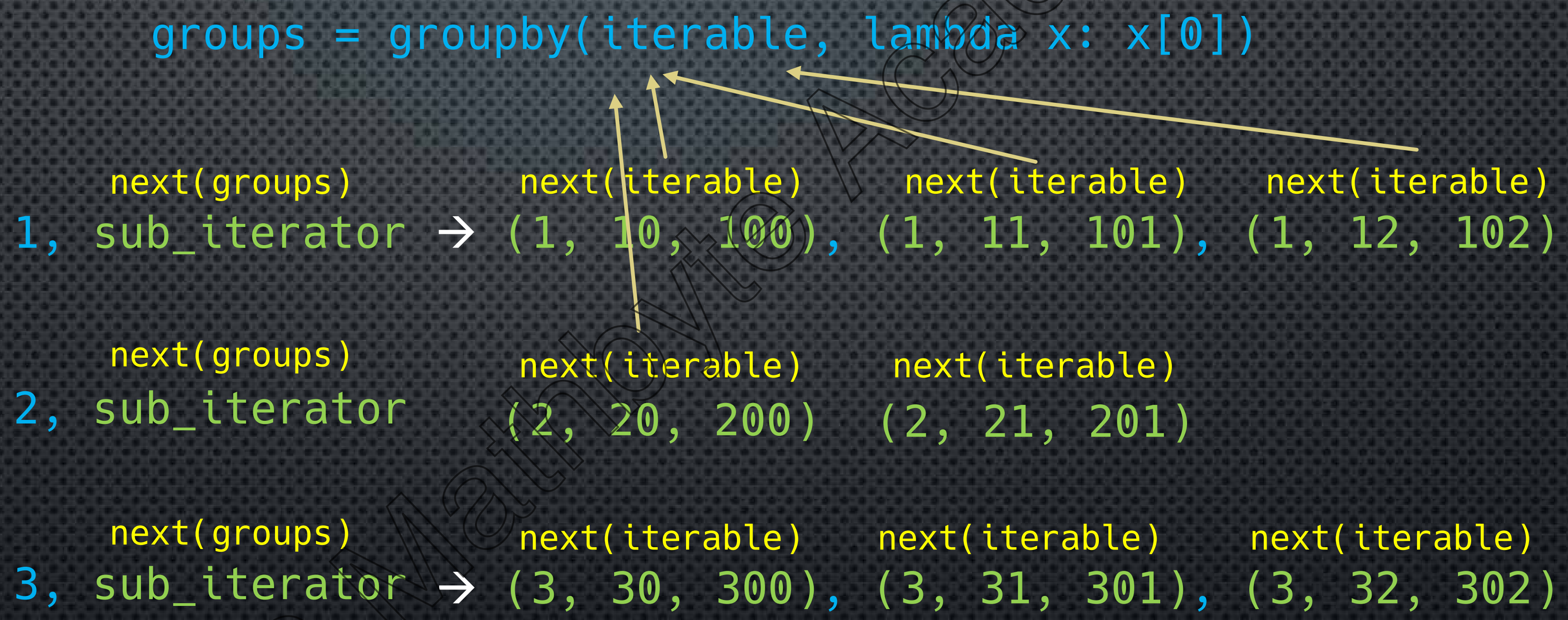
```
1, sub_iterator → (1, 10, 100), (1, 11, 101), (1, 12, 102)
2, sub_iterator → (2, 20, 200), (2, 21, 201)
3, sub_iterator → (3, 30, 300), (3, 31, 301), (3, 32, 302)
```

note how the sequence is sorted by the grouping key!

Important Note

The sequence of elements produced from the "sub-iterators" are all produced from the same underlying iterator

```
iterable
(1, 10, 100)
(1, 11, 101)
(1, 12, 102)
(2, 20, 200)
(2, 21, 201)
(3, 30, 300)
(3, 31, 301)
(3, 32, 302)
```



```
groups = groupby(iterable, lambda x: x[0])

   next(groups)         next(iterable)  next(iterable)  next(iterable)
1, sub_iterator  →  (1, 10, 100), (1, 11, 101), (1, 12, 102)

   next(groups)         next(iterable)  next(iterable)
2, sub_iterator     (2, 20, 200)  (2, 21, 201)

   next(groups)         next(iterable)  next(iterable)  next(iterable)
3, sub_iterator  →  (3, 30, 300), (3, 31, 301), (3, 32, 302)
```

`next(groups)` actually iterates through all the elements of the current "sub-iterator" before proceeding to the next group

Coding Exercises