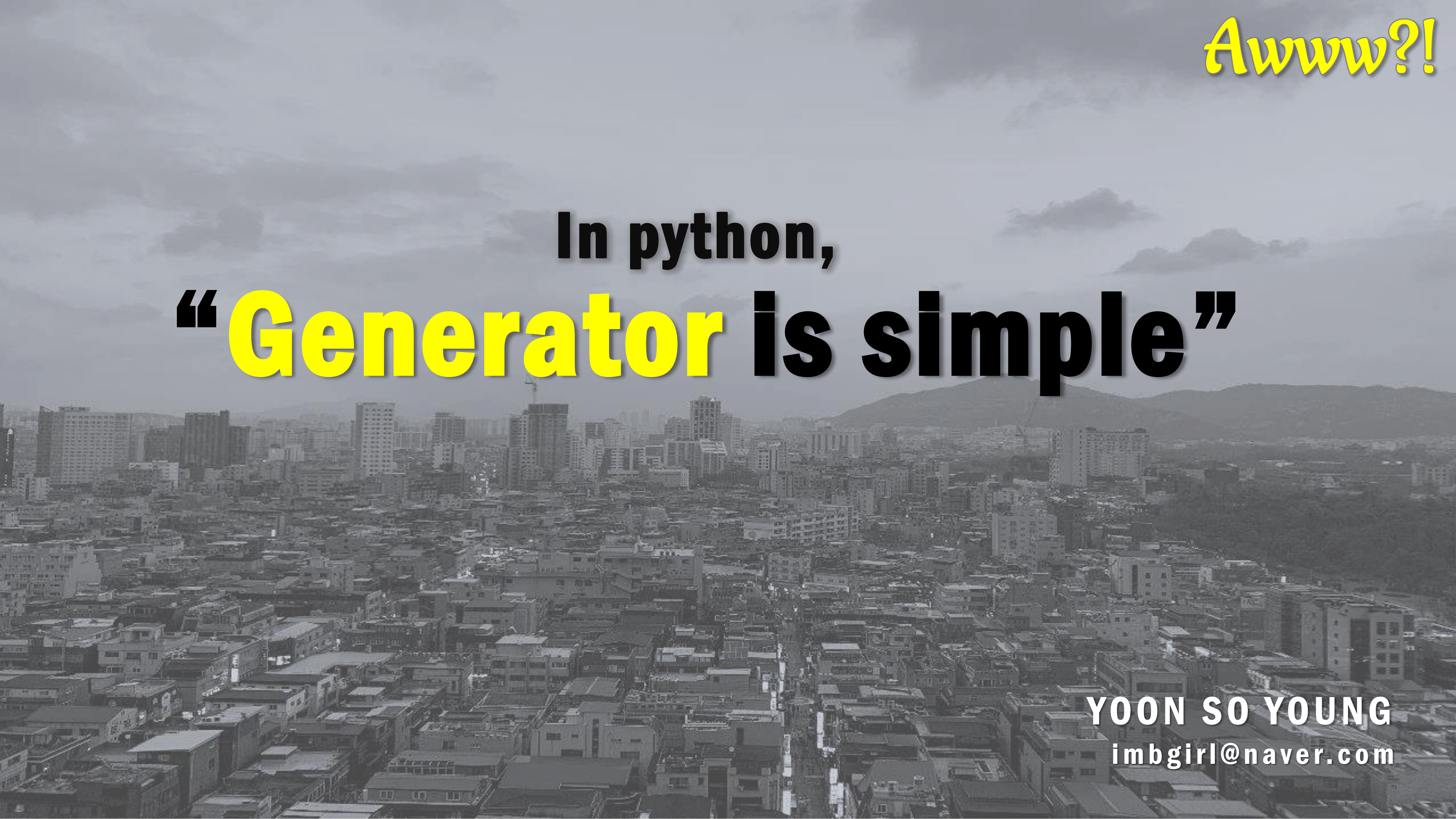
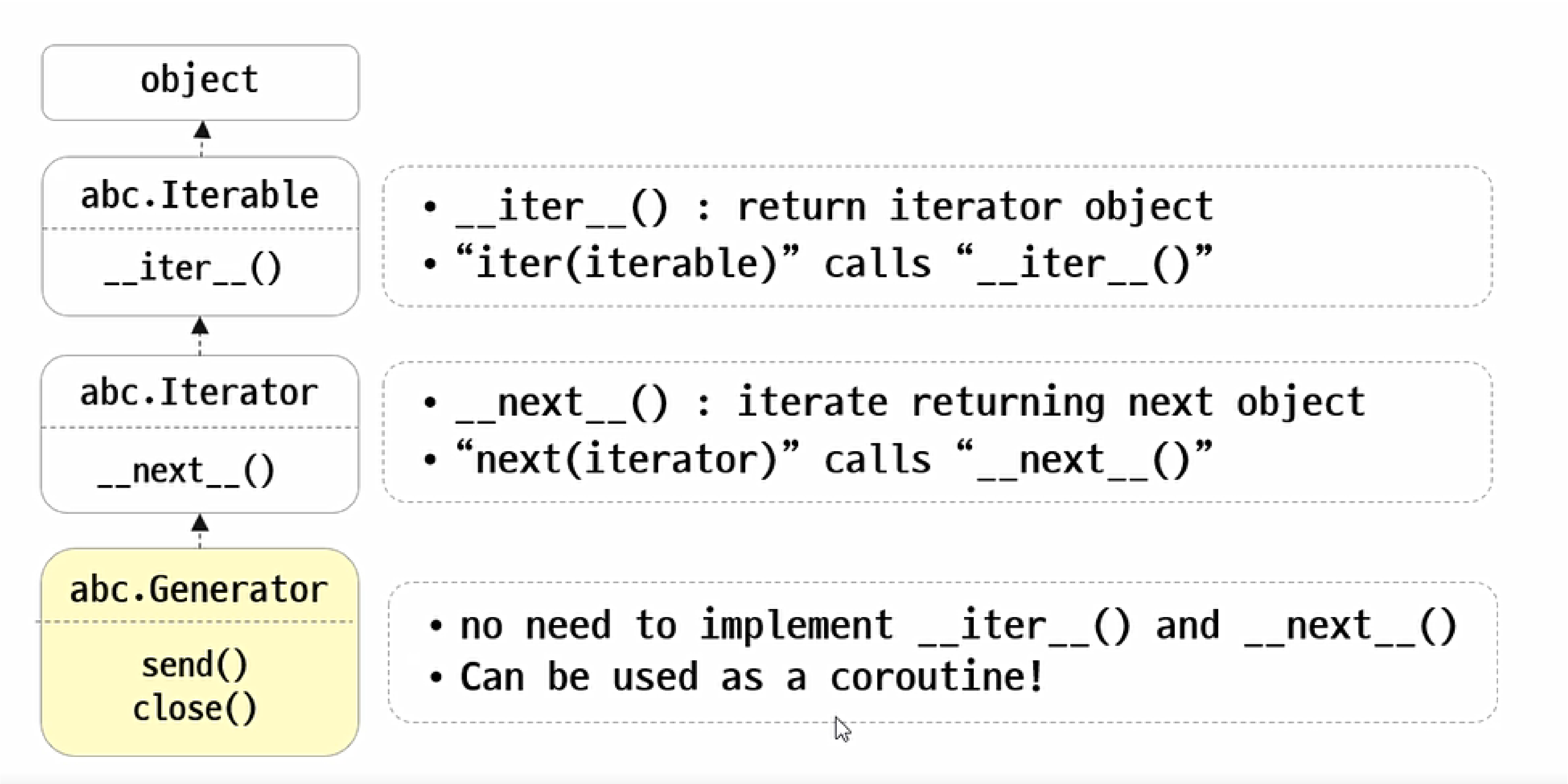


Awww?!

In python,
“Generator is simple”

YOON SO YOUNG
imbgirl@naver.com

‘Generator 1/2’



- __iter__() : return iterator object
- “iter(iterable)” calls “__iter__()”

- __next__() : iterate returning next object
- “next(iterator)” calls “__next__()”

- no need to implement __iter__() and __next__()
- Can be used as a coroutine!

‘Generator 2/3’

- How generators work like?
  - ✓ Generate produce a value and suspend by next()
  - ✓ “StopIteration” occurs when the produce is ended

next(generator) ➔ an object

- What generators are for?

Generate elements, one at the time, and suspend ...

1. Save memory
2. Support iteration pattern, infinite sequences, etc.

‘Generator 2/3’

Awww?!

abc.Iterator

- ✓ need to implement ‘__iter__()’ and ‘__next__()’
- ✓ get from ‘iter(iterable)’

abc.Generator

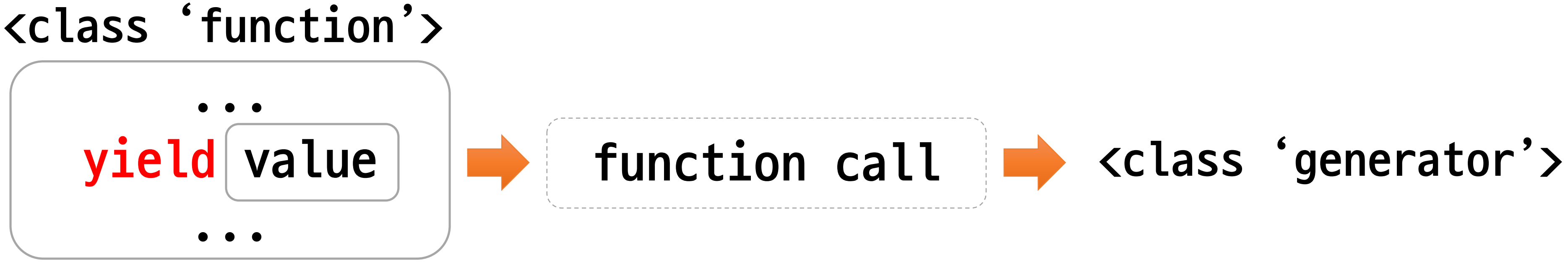
- ✓ no need to implement ‘__iter__()’ and ‘__next__()’

‘Generator 3/3’

Awww?!

- How to make a generator?

1. Make function with the keyword ‘yield’
2. Generator Expression



<class ‘function’>

```
...
yield value
...
```

function call

<class ‘generator’>

<class ‘generator’>

```
( value for target in iterable [ if ... ] )
```