



Python 3 Beginner

Chapter 00

Previously, on Python 3 Beginner...

release 1.0.0

What we saw last time

- ➡ The `NoneType` datatype and its only value: `None`
- ➡ The `bool` type and its 2 values: `True` and `False`
- ➡ Comparing values with operators: `==`, `!=`, `<`, `in...`
- ➡ Combining `bool` values with logical operators: `and` and `or`
- ➡ Conditionally running code with `if`, `else`, `elif`
- ➡ Iterating over sequences with `for`
- ➡ Looping on a condition with `while`
- ➡ Short-circuiting loops with `break` and `continue`



Q&A

Where? Who?

Class Material

► GitHub: <https://github.com/cstar-industries/python-3-beginner>

Instructor

► Charles Francoise <charles@cstar.io>



Session Objectives

At the end of this session, you will be able to:

- ➡ Master the `list` type
- ➡ Use `str` to its full potential
- ➡ Use and recognize the `tuple` type for immutable ordered lists
- ➡ Use the `set` type to perform efficient inclusion tests and set arithmetic
- ➡ Use and master the `dict` type to associate arbitrary keys and values
- ➡ Wield Python superpowers like list comprehensions and more...



Session Syllabus

Chapter	Subject	Start	End	Total Time (in hours)
00	Previously on Python 3 Beginner...	14:00	14:30	00:30
01	More about list	14:30	15:00	00:30
02	Other sequence types: str – tuple – set	15:00	15:45	00:45
	<i>Coffee Break</i>	15:45	16:00	00:15
03	Associative container: dict	16:00	16:30	00:30
04	Mastering container types	16:30	17:00	00:30
	<i>Coffee Break</i>	17:00	17:15	00:15
	Workshop	17:15	18:00	00:45
	Total			04:00



Python 3 Beginner

Chapter **01**

More about list

release 1.0.0

list: what we already know

➡ Test inclusion with `in` and `not in`

```
'toto' in [0, 3.33, False, 'toto', []]
```

True

➡ Extend with `+` and `*`

```
[1, 2] + [3]
```

[1, 2, 3]

```
3 * [1, 2]
```

[1, 2, 1, 2, 1, 2]

list: what we already know

➡ Get list length with len()

```
len(['Hello', 'World', '!'])
```

```
3
```

➡ Extract items with []

```
l = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
print(l[2])
print(l[:2])
print(l[1:8:2])
```

```
2
[0, 1]
[1, 3, 5, 7]
```

list: methods and built-in functions

➡ Find the first occurrence of an item: `l.index`

```
l = ['H', 'e', 'l', 'l', 'o']
l.index('l')
```

```
2
```

➡ Return the number of occurrences of an item: `l.count`

```
print(l.count('l'))
print(l.count('J'))
```

```
2
0
```

list: methods and built-in functions

► Get the largest and smallest items: `max` and `min`

```
l = [9, 2, 8, 5, 0, 1, 4, 7, 6, 3]  
print(min(l), max(l))
```

0 9

► Add all items together: `sum`

```
sum([1, 2, 5, 10])
```

18

Let's write some code!

```
from random import shuffle

l = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
shuffle(l)

print(l)
print(f'max value: {max(l)}, min value: {min(l)}')
```

```
[3, 6, 0, 1, 5, 7, 8, 9, 2, 4]
max value: 9, min value: 0
```

list is a **mutable** object

➡ Replace items at indexes or ranges

```
l = [0, 1, 2, 3, 4]
l[1] = 'toto'
print(l)
```

```
[0, 'toto', 2, 3, 4]
```

```
l[2:4] = [8, '8', 'eight']
print(l)
```

```
[0, 'toto', 8, '8', 'eight', 4]
```

```
l[1:5] = []
print(l)
```

```
[0, 4]
```

list is a **mutable** object

➡ Delete items at indexes or ranges

```
l = [0, 1, 2, 3, 4]  
del l[1]  
print(l)
```

```
[0, 2, 3, 4]
```

```
del l[:2]  
print(l)
```

```
[3, 4]
```

list is a **mutable** object

► Add and remove items at the end of the list

```
l = [0, 1, 2, 3, 4]
l.append(5)
print(l)
```

```
[0, 1, 2, 3, 4, 5]
```

```
a = l.pop()
print(l)
print(a)
```

```
[0, 1, 2, 3, 4]
5
```

list is a **mutable** object

➡ Add and remove items at an arbitrary position

```
l = [0, 1, 2, 3, 4]
l.insert(1, 5)
print(l)
```

```
[0, 5, 1, 2, 3, 4]
```

```
a = l.pop(3)
print(l)
print(a)
```

```
[0, 5, 1, 3, 4]
2
```

list is a **mutable** object

► Remove the first occurrence of an item

```
l = ['H', 'e', 'l', 'l', 'o']
l.remove('l')
print(l)
```

```
['H', 'e', 'l', 'o']
```

► Copy a list

```
a = [0, 1, 2, 3, 4]
b = a.copy()
print(b)
print(a is b)
```

```
[0, 1, 2, 3, 4]
False
```

list is a **mutable** object

➡ Remove all elements from a list

```
l = ['H', 'e', 'l', 'l', 'o']
print(l)

l.clear()
print(l)
```

```
['H', 'e', 'l', 'l', 'o']
[]
```

Let's write some code!

```
a = [0, 1, 2, 3, 4, 5]
b = a.copy()

print(a)
print(b)
print(a == b)
print(a is b)
```

```
[0, 1, 2, 3, 4, 5]
[0, 1, 2, 3, 4, 5]
True
False
```

⚠️ Mutability can be *dangerous* ⚠️

➡️ If an object is mutable, changing it in one place also changes it in another.

```
a = [0, 1, 2, 3, 4]
b = a

last_item_of_b = b.pop()
print(b)

print(a)
```

```
[0, 1, 2, 3]
[0, 1, 2, 3]
```

⚠️ Mutability can be *dangerous* ⚠️

➡ Whenever possible, use `copy` to preserve the original object.

```
a = ['Hello', 'hello', 'hi']
b = a.copy()

print(a == b)
print(a is b)

a.pop()
print(b)
```

```
True
False
['Hello', 'hello', 'hi']
```

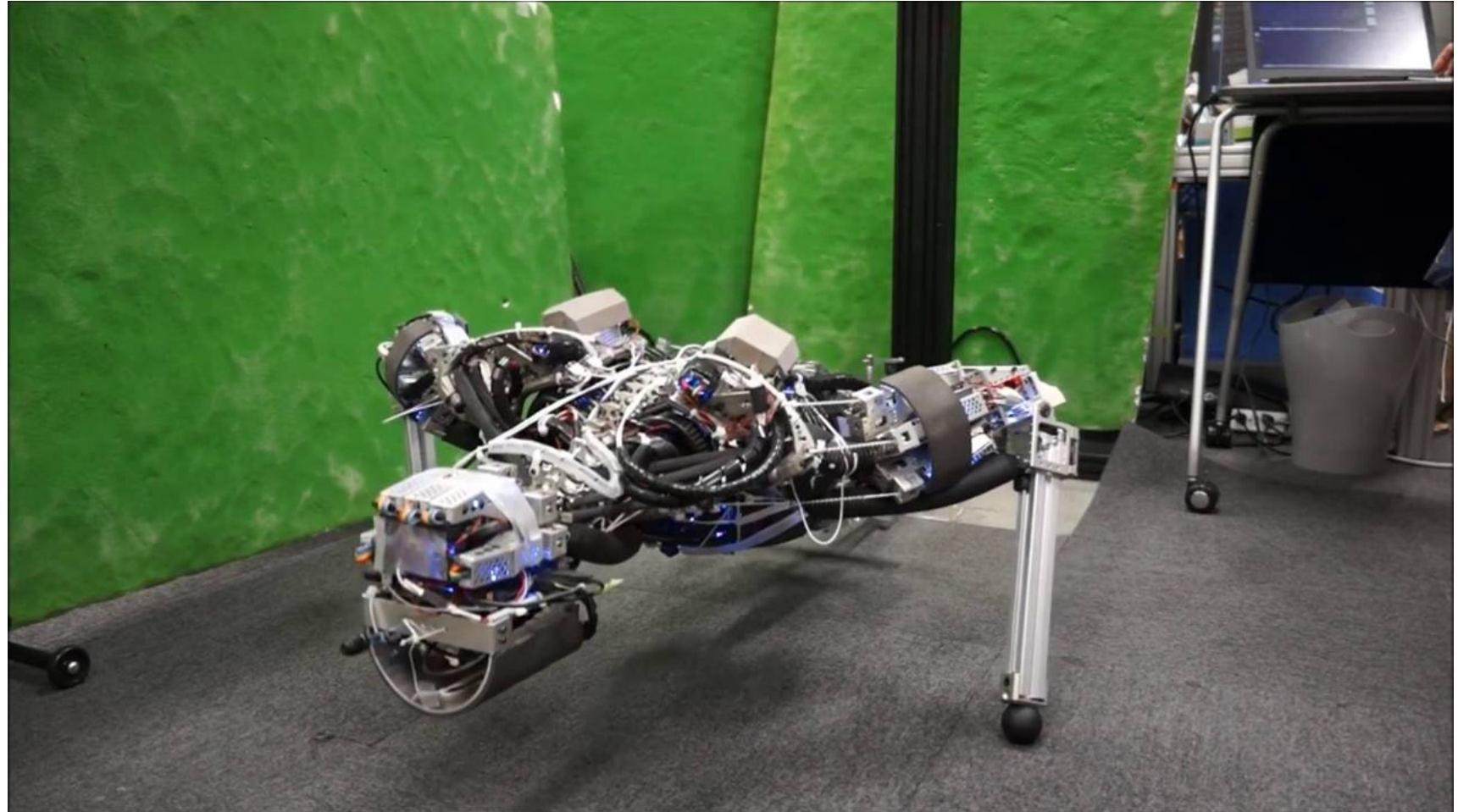
⚠️ Mutability can be *dangerous* ⚠️

➡ Beware! copy is *shallow*

```
a = [[‘a’, ‘b’], [‘a’, ‘c’], [‘b’, ‘c’]]  
b = a.copy()  
  
print(a == b)  
print(a is b)  
print(a[0] is b[0])  
  
b[0][0] = None  
print(a)
```

```
True  
False  
True  
[[None, ‘b’], [‘a’, ‘c’], [‘b’, ‘c’]]
```

Workout Time!

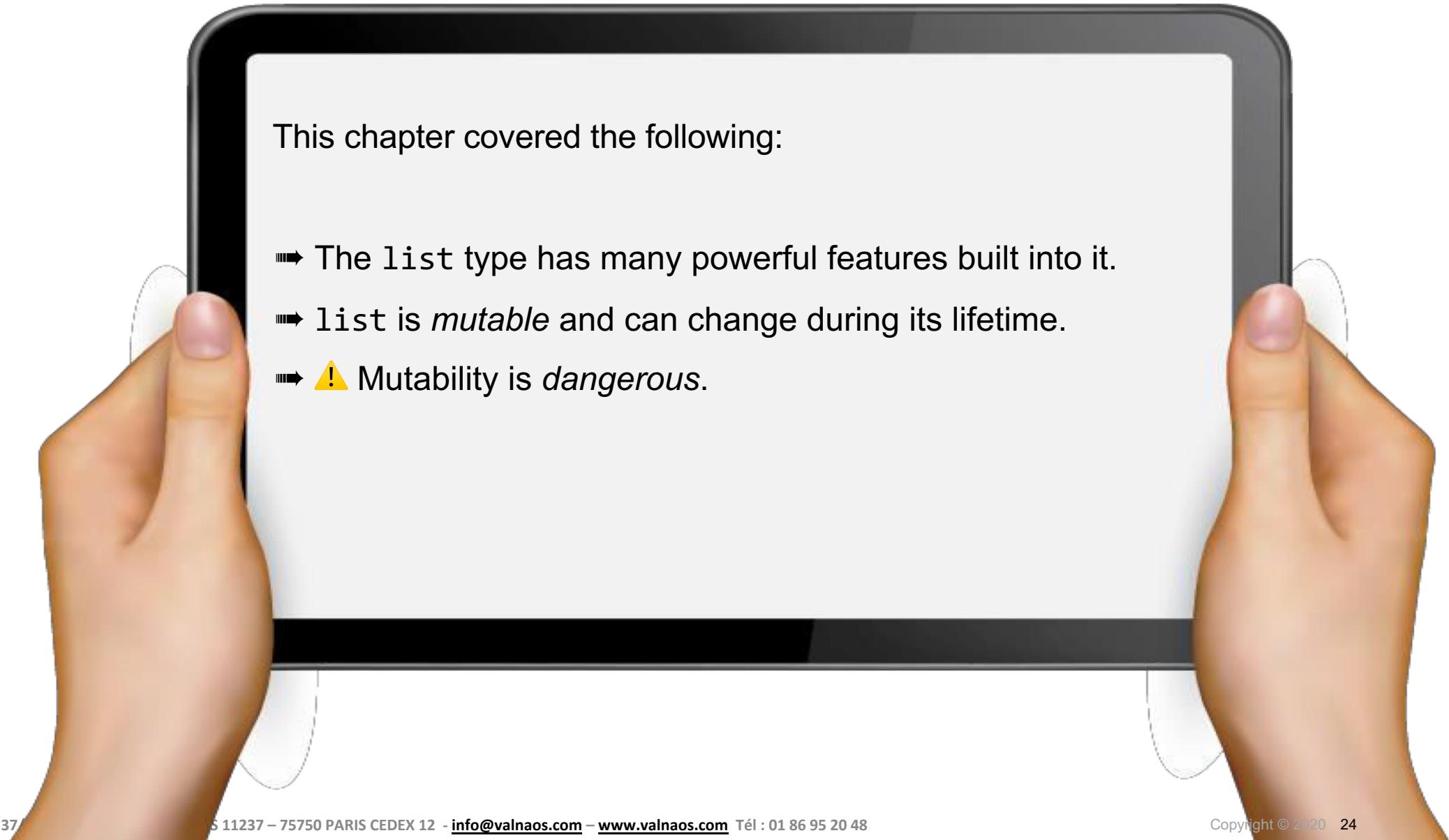


[Chapter 1 Exercises](#) link

Chapter Summary

This chapter covered the following:

- ➡ The `list` type has many powerful features built into it.
- ➡ `list` is *mutable* and can change during its lifetime.
- ➡ ⚡ Mutability is *dangerous*.





Python 3 Beginner

Chapter **02**

More sequence datatypes

release 1.0.0

An immutable sequence type: tuple

- ➡ A tuple is very similar to a list, but it is *immutable*
- ➡ It is most often written as a comma-separated series of values between parentheses

```
t = (0, 1, 2)
print(t)
print(type(t))
```

```
(0, 1, 2)
tuple
```

An immutable sequence type: tuple

➡ Parentheses are optional

```
t = 0, 1, 2
print(t)
print(type(t))
```

```
(0, 1, 2)
tuple
```

An immutable sequence type: tuple

- Use a comma for a single-item tuple

```
t = (0,)  
print(t)
```

(0,)

- Use empty parentheses for an empty tuple

```
t = ()  
print(t)
```

()

An immutable sequence type: tuple

- tuples can be used to destructure other sequences at assignment

```
(a, b, c) = range(3)  
print(a)  
print(b)  
print(c)
```

```
0  
1  
2
```

- You've already met tuple before...

```
for i, x in enumerate(l):  
    ...
```

tuple is like list

➡ Iteration with for

```
for x in t:  
    ...
```

➡ Indexation with []

```
t[0]  
t[-2:]
```

➡ Inclusion with in

```
0 in (0, 1, 2)  
1138 not in (3,)
```

tuple is *unlike* list

► tuple is immutable

```
t = (0, 1, 2)  
t[0] = 'toto'
```

TypeError: 'tuple' object does not support item assignment

```
t.pop()
```

AttributeError: 'tuple' object has no attribute 'pop'

Let's write some code!

```
t = (0, 1, 2)
print(t)
print(type(t))
```

```
(0, 1, 2)
<class 'tuple'>
```

More about str

► Match prefix and suffix

```
filename = 'sensor_data.csv'  
print(filename.startswith('Sens'))  
print(filename.endswith('.csv'))
```

True

False

► Find occurrence of a substring inside a string

```
print(filename.find('data'))  
print(filename.find('hello'))
```

7

-1

More about str

- ➡ Quickly validate the content of a string
 - ➡ `isalpha`: checks if all characters are letters
 - ➡ `isnumeric`, `isdigit`, `isdecimal`: checks if all characters are numeric, digits, decimal digits, respectively
 - ➡ `isalnum`: checks if all characters are either letter or numeric
 - ➡ `isupper`, `islower`: checks if all character are upper or lowercase
 - ➡ `isspace`: checks if all characters are whitespace

More about str

```
s = 'hello'  
print(s.isalpha())  
print(s.isnumeric())  
print(s.isalnum())  
print(s.isupper())  
print(s.islower())
```

```
True  
False  
True  
False  
True
```

More about str

► Split strings into a list of strings, separated by a delimiter

```
languages = 'C,C++,Java,Python,Go'  
data.split(',')
```

```
[ 'C', 'C++', 'Java', 'Python', 'Go' ]
```

► Join a list of strings using a delimiter

```
languages = '\n'.join(languages)  
print(languages)
```

```
C  
C++  
Java  
Python  
Go
```

More about str

► Replace all occurrences of a substring in a string by another

```
languages = 'C,C++,Java,Python,Go'  
languages.replace(',', '\n')  
print(languages)
```

```
C  
C++  
Java  
Python  
Go
```

Let's write some code!

```
filename = 'toto.txt'  
print(filename.startswith('hello'))  
print(filename.endswith('.txt'))
```

False
True

set

➡ An **unordered** set of **distinct** objects

```
s = set([1, 2, 3])  
print(s)
```

```
{1, 2, 3}
```

```
s1 = {3, 2, 1}  
print(s1)  
print(s == s1)
```

```
{1, 2, 3}  
True
```

```
print({1, 1, 1, 1, 1, 1})
```

```
{1}
```

set

► Test inclusion

```
s = {1, 2, 3}  
print(1 in s)  
print(0 in s)
```

True
False

► No indexing

```
s[0]
```

`TypeError: 'set' object does not support indexing`

set

➡ Iteration

```
s = set('Python')
for c in s:
    print(c)
```

```
P
t
o
y
n
h
```

set is mutable

➡ add an item to the set

```
s = set()  
s.add(1)  
print(s)
```

{1}

```
s = {1, 2, 3}  
s.add(1)  
print(s)
```

{1, 2, 3}

set is mutable

➡ remove an item from the set

```
s = {1, 2, 3}  
s.remove(1)  
print(s)
```

```
{2, 3}
```

```
s.remove('a')
```

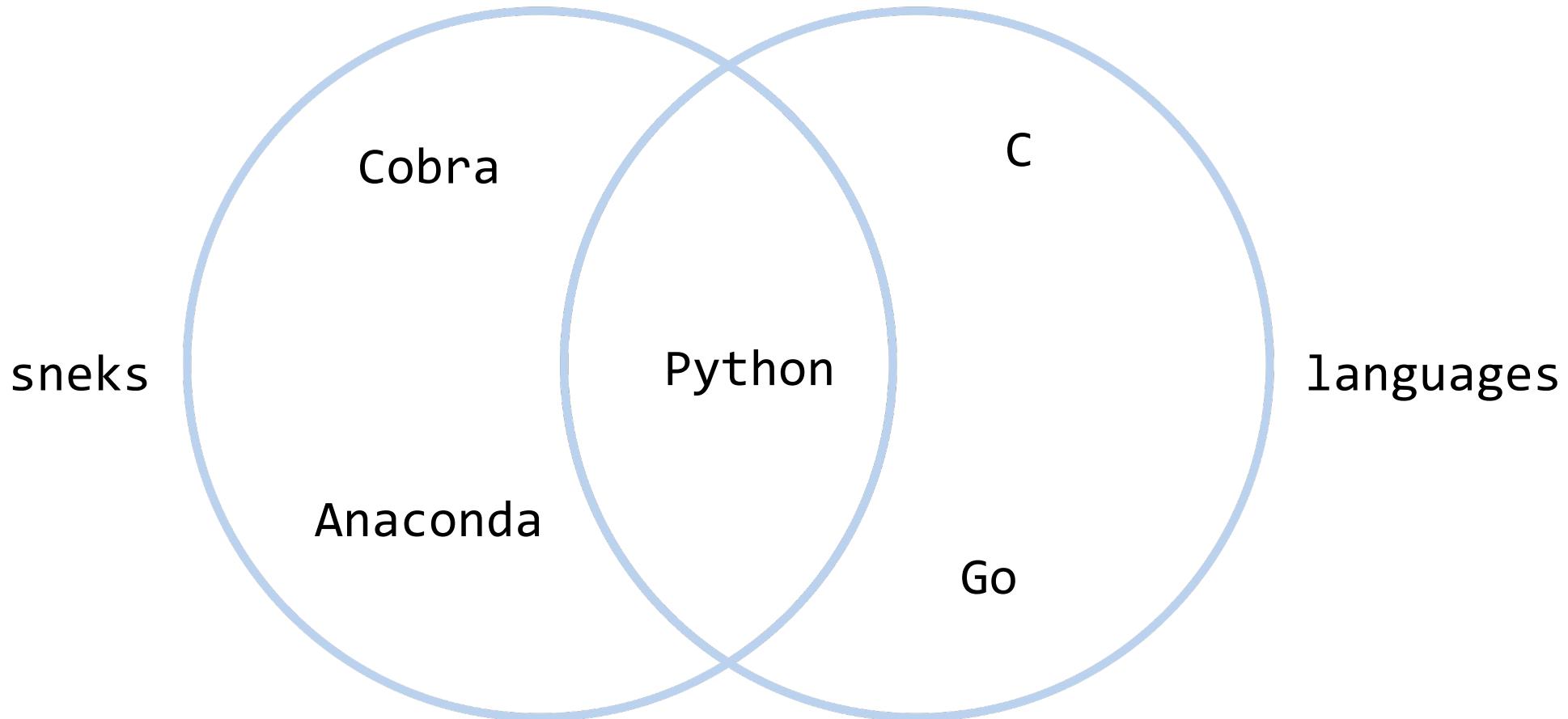
```
KeyError: 'a'
```

➡ remove all items from the set

```
s.clear()  
print(s)
```

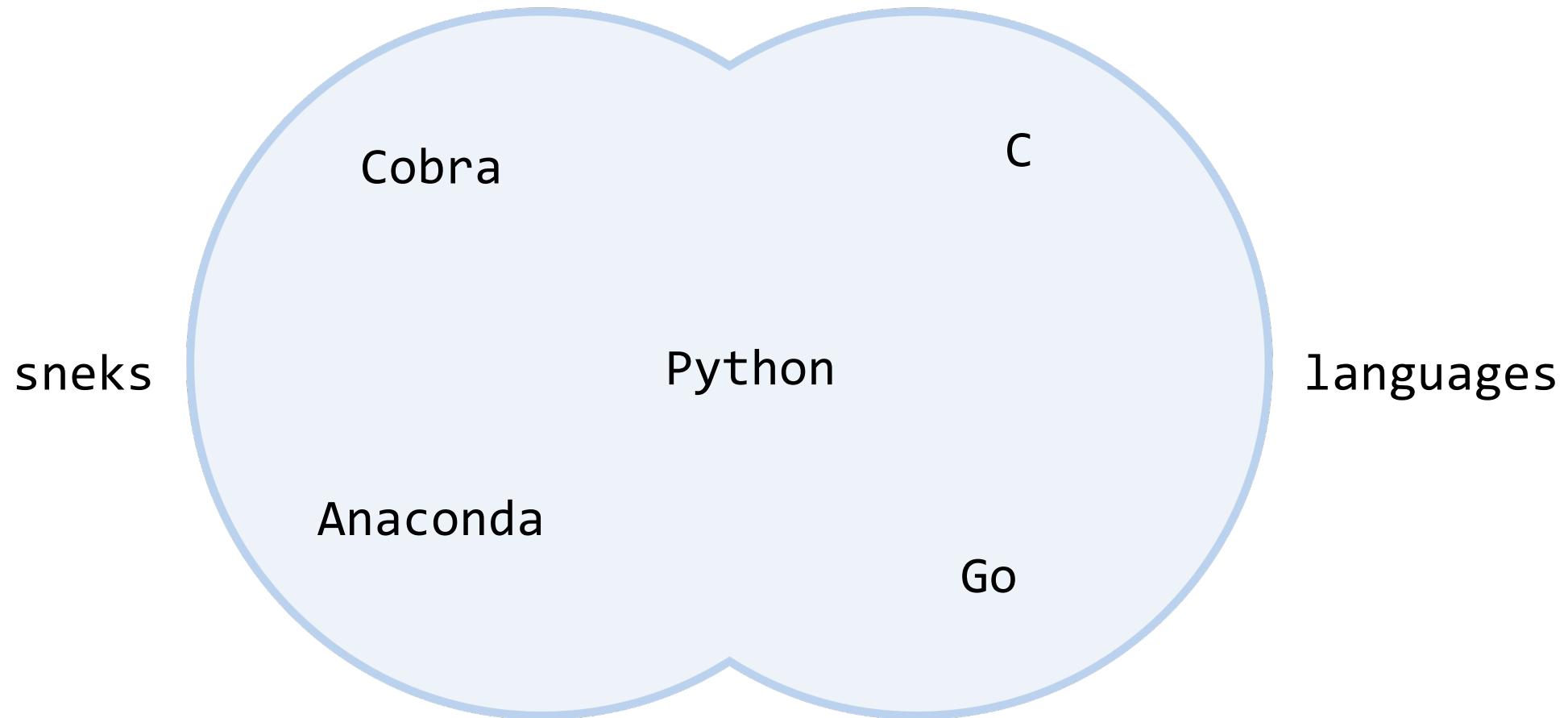
```
set()
```

Set arithmetic

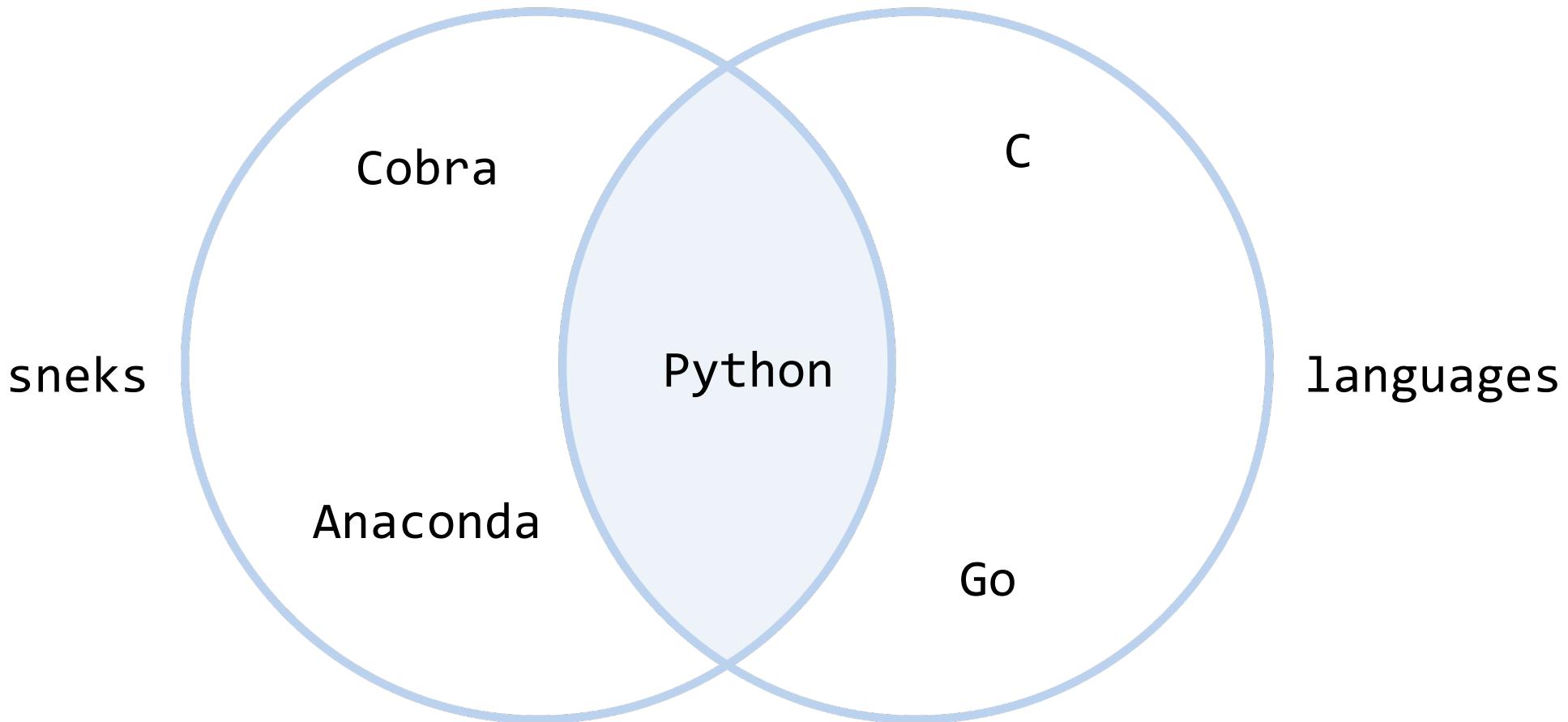


```
sneks = {'Python', 'Cobra', 'Anaconda'}
languages = {'Python', 'C', 'Go'}
```

Union "OR"



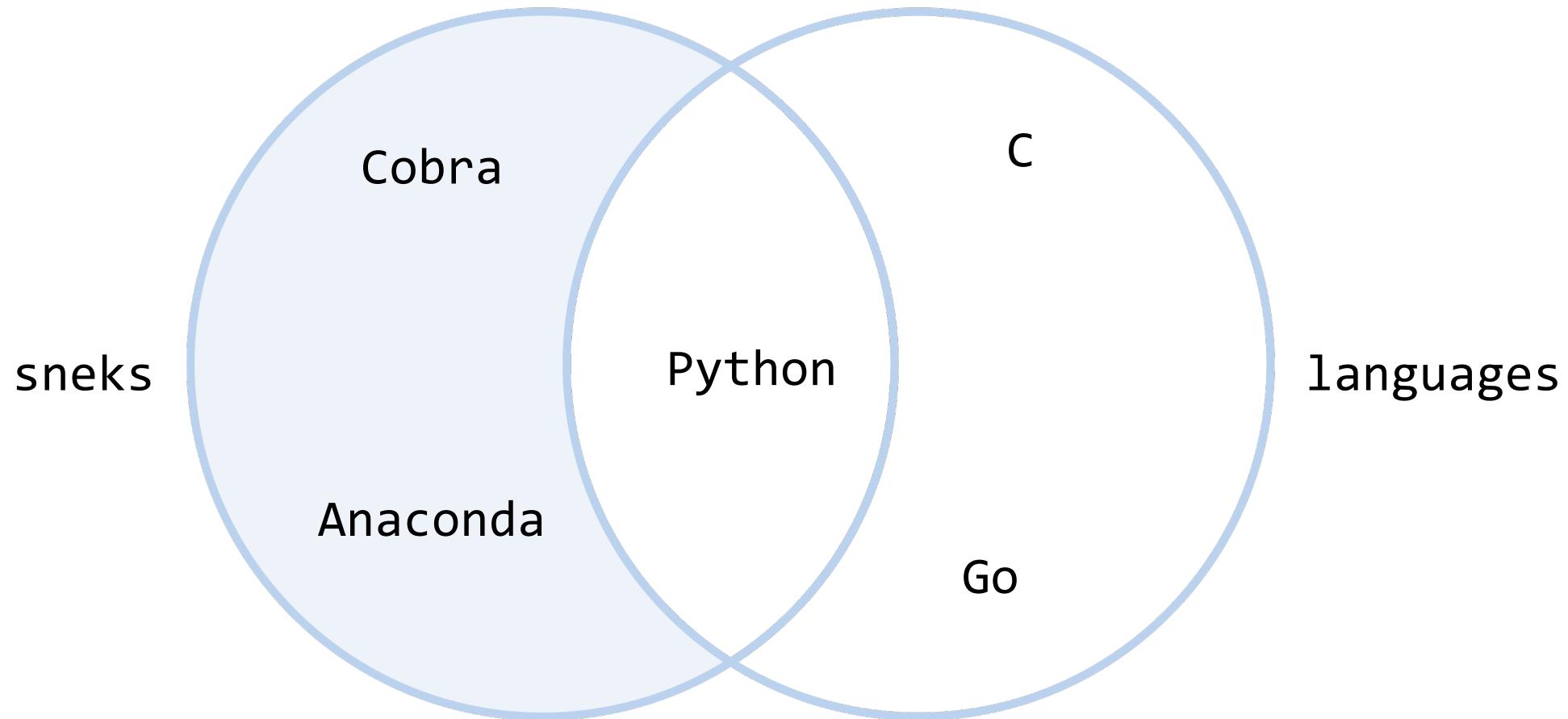
Intersection "AND"



sneks & languages

{ 'Python' }

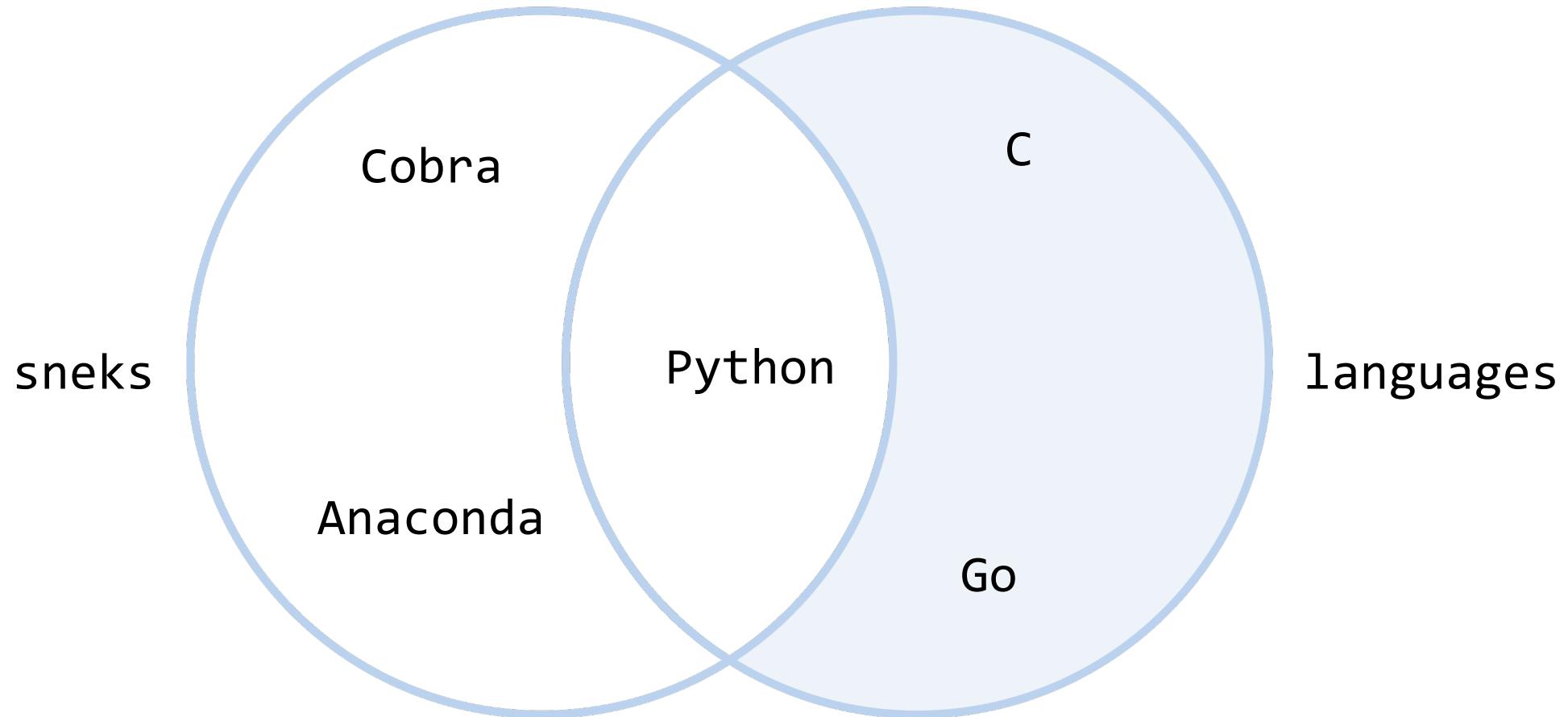
Difference



sneks - languages

{ 'Anaconda' , 'Cobra' }

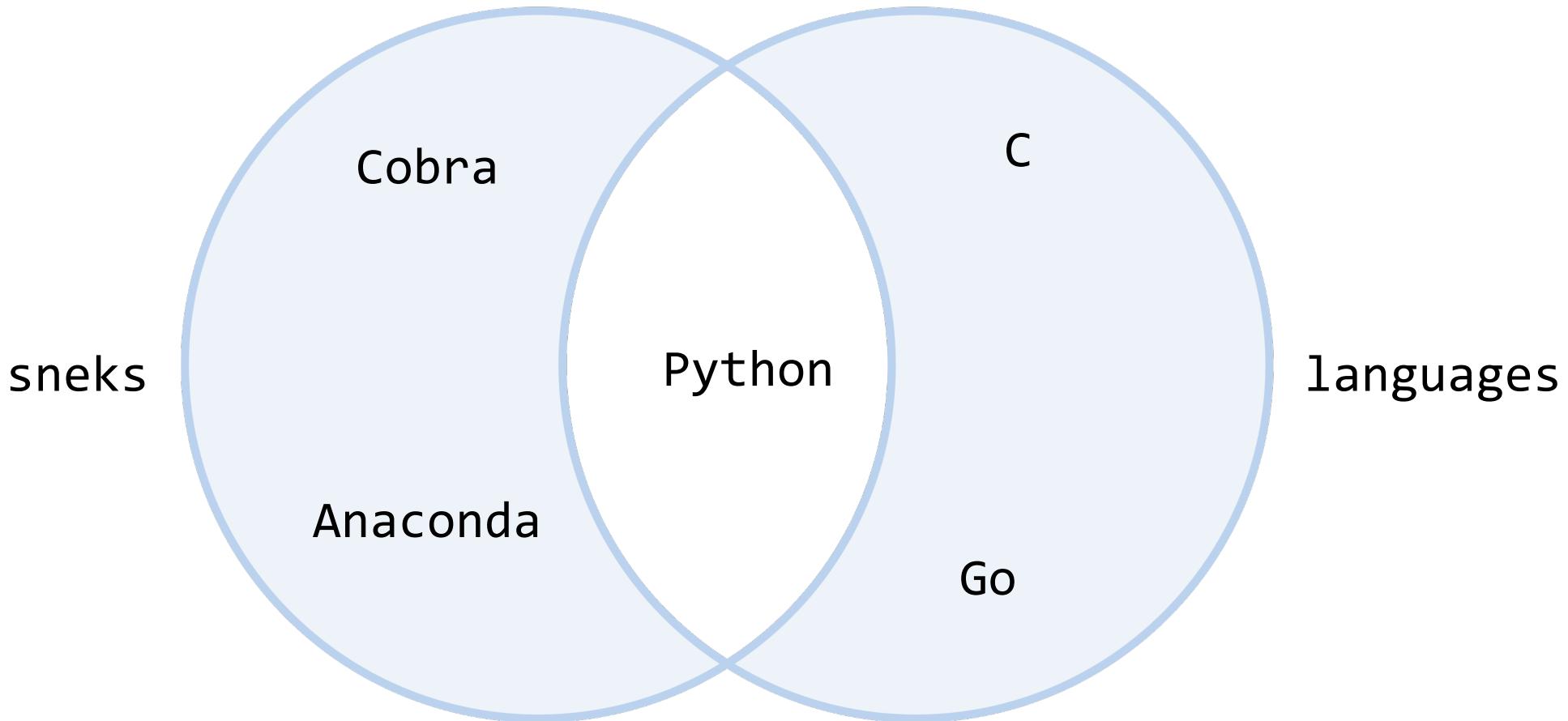
Difference



languages - sneks

{ 'C', 'Go' }

Exclusive disjunction "XOR"



```
languages ^ sneks
```

```
{'C', 'Cobra', 'Anaconda', 'Go'}
```

set is mutable

➡ Use assignment operators to mutate

```
languages |= {'C++', 'Java'}  
print(languages)
```

```
{'Go', 'C', 'Python', 'Java', 'C+'}
```

➡ ! Watch out for mutability!

```
s = languages  
print(s is languages)  
s &= sneks  
print(s)  
print(languages)
```

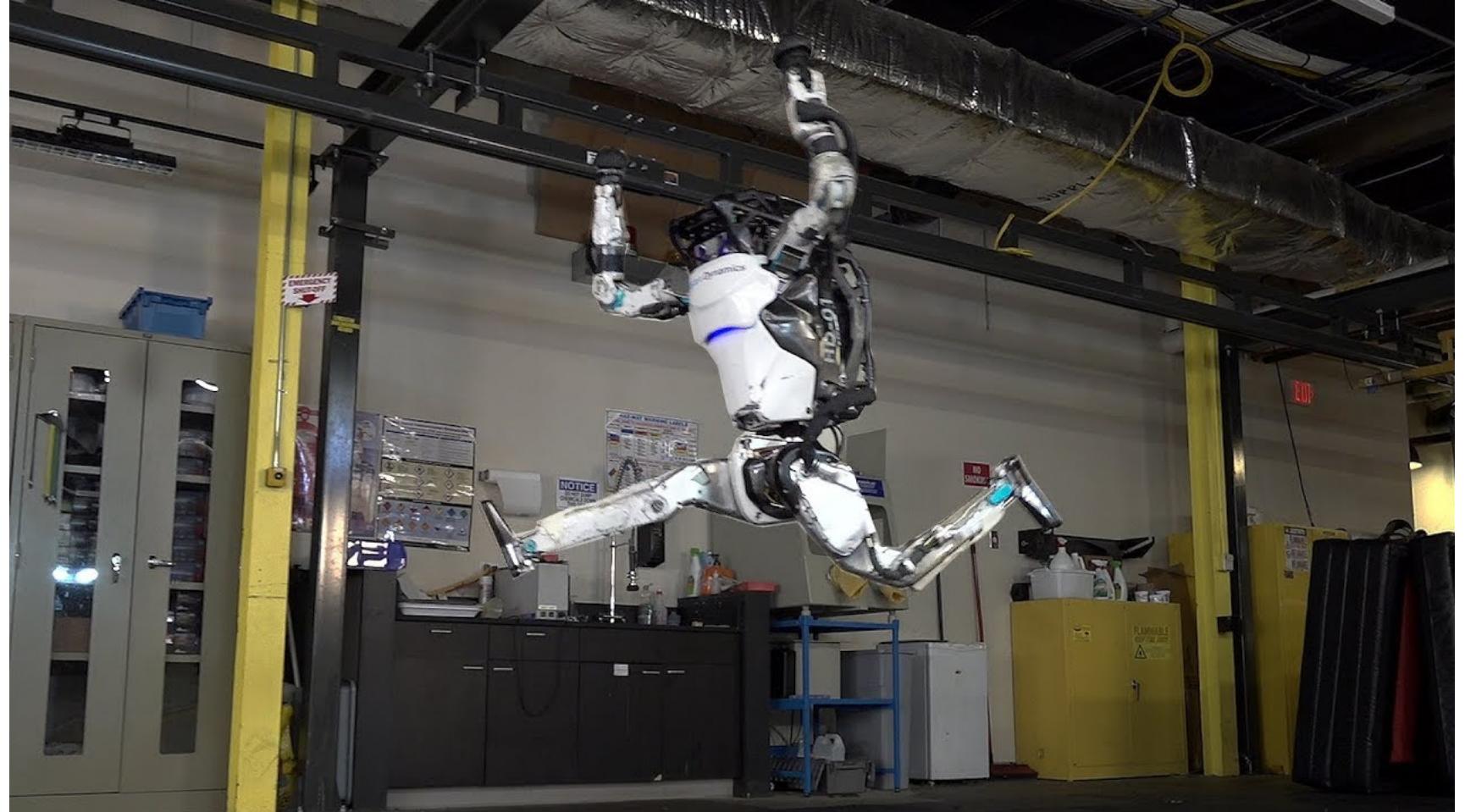
```
True  
{'Python'}  
{'Python'}
```

Let's write some code!

```
print('Union:', sneks | languages)
print('Intersection:', sneks & languages)
print('Left difference:', sneks - languages)
print('Right difference:', languages - sneks)
print('Exclusion:', languages ^ sneks)
print('Exclusion:', (languages | sneks) - (languages & sneks))
```

```
Union: {'Cobra', 'Anaconda', 'Go', 'C', 'Python'}
Intersection: {'Python'}
Left difference: {'Anaconda', 'Cobra'}
Right difference: {'C', 'Go'}
Exclusion: {'C', 'Cobra', 'Anaconda', 'Go'}
Exclusion: {'Anaconda', 'Cobra', 'C', 'Go'}
```

Workout Time!

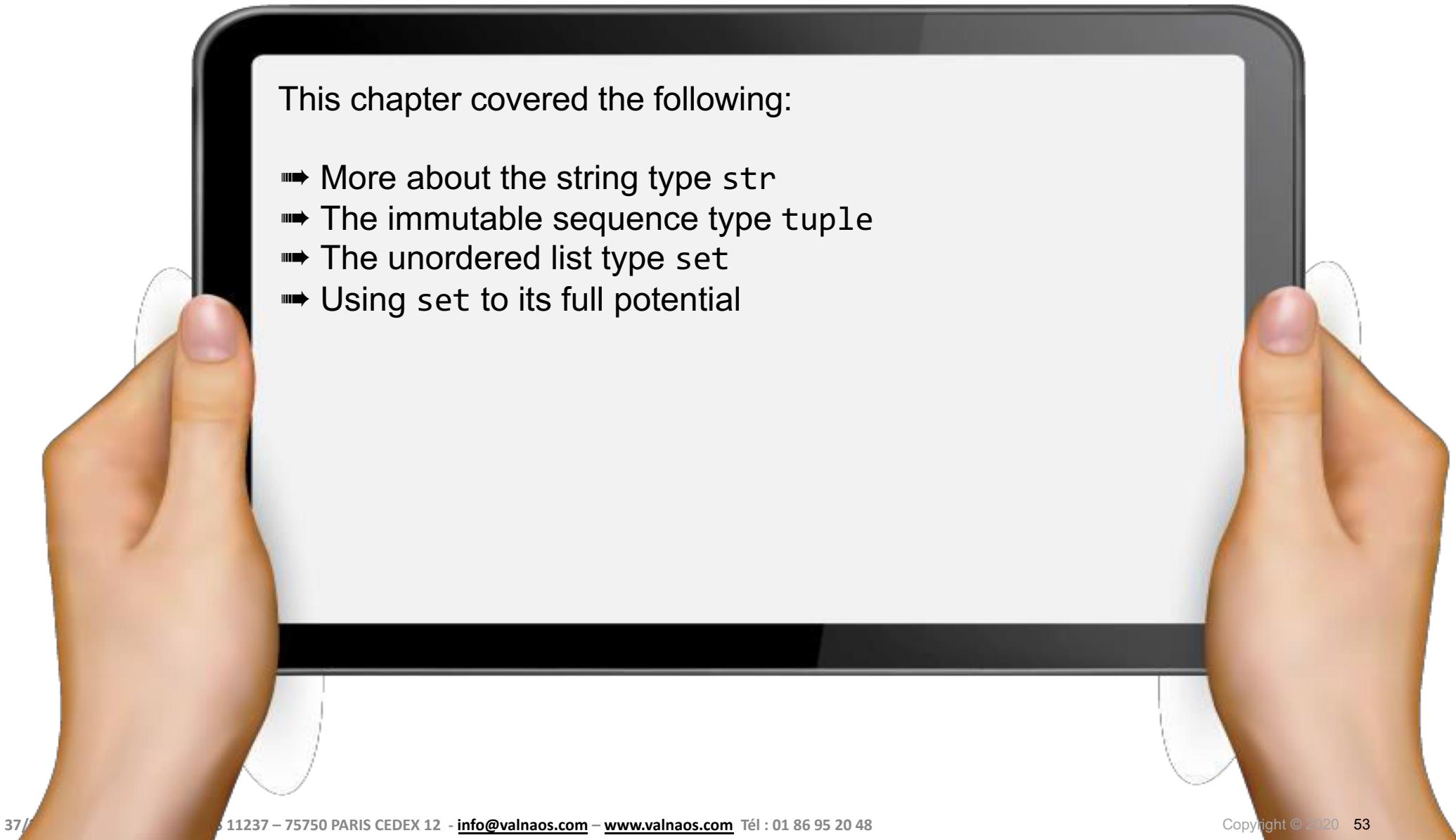


[Chapter 2 Exercises](#) link

Chapter Summary

This chapter covered the following:

- ➡ More about the string type `str`
- ➡ The immutable sequence type `tuple`
- ➡ The unordered list type `set`
- ➡ Using `set` to its full potential





Python 3 Beginner

Chapter **03**

Associative container: dict

release 1.0.0

An associative container: dict

- ➡ Maps *keys* to *values*
- ➡ Values can be any object
- ➡ Keys can be *almost* any object (must be *hashable*)
- ➡ Similar to C++ std::map, Java HashMap, Ruby hashes, JavaScript objects...
- ➡ If you've used JSON, you've used dicts

dict basics

➡ Creating a dictionary

```
d = {0: 'zero', 2: 'two', 1: 'one'}  
type(d)
```

dict

```
d1 = {'one': 1, 'two': '2', 'three': 3.0, 'four': [4, 4, 4]}  
print(d1)
```

{'one': 1, 'two': '2', 'three': 3.0, 'four': [4, 4, 4]}

dict basics

► Dictionary creation & equality

```
d1 = {'one': 1, 'two': '2', 'three': 3.0, 'four': [4, 4, 4]}  
print(d1)
```

```
{'one': 1, 'two': '2', 'three': 3.0, 'four': [4, 4, 4]}
```

```
d2 = dict(one=1, four=[4, 4, 4], two='2', three=3.0)  
print(d2)  
print(d1 == d2)
```

```
{'one': 1, 'four': [4, 4, 4], 'two': '2', 'three': 3.0}  
True
```

```
d2 = dict([('one', 1), ('four', [4, 4, 4]), ('two', '2'), ('three', 3.0)])  
print(d3 == d1)
```

```
True
```

dict basics

➡ Retrieving an item

```
d = {0: 'zero', 2: 'two', 1: 'one'}  
d[2]
```

'two'

```
d['two']
```

KeyError: 'two'

```
print(d.get('two'))
```

None

Iterating dict

➡ By keys

```
for k in d:  
    print(f'{k}=>{d[k]}', end=' ')
```

```
2=>two 1=>one 4=>four
```

➡ By values

```
for v in d.values():  
    print(v, end=' ')
```

```
two one four
```

➡ By key-value pairs

```
for k, v in d.items():  
    print(f'{k}=>{v}', end=' ')
```

```
2=>two 1=>one 4=>four
```

Let's write some code!

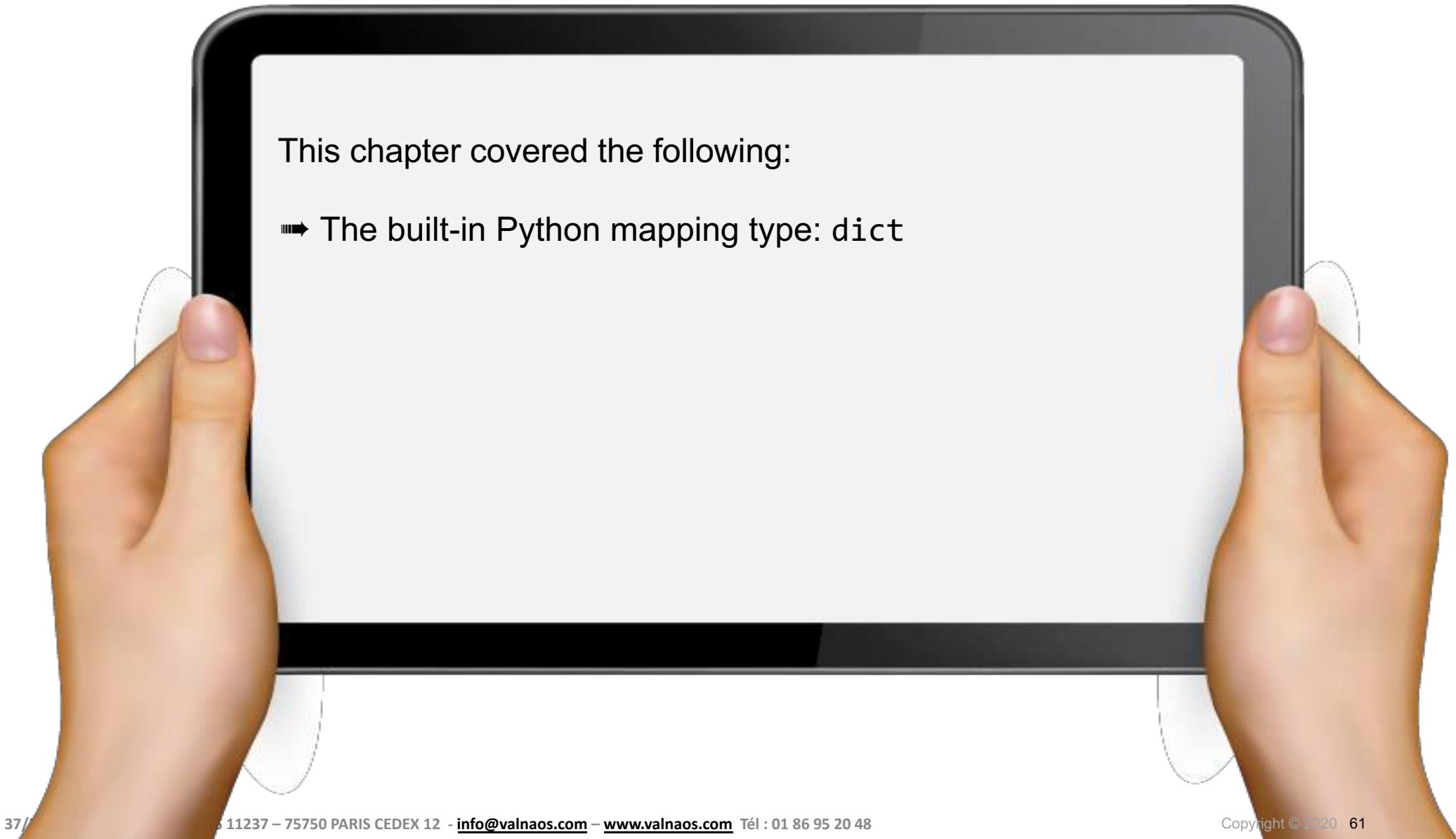
```
for k, v in d.items():
    print(f'{k}>{v}', end=' ')
```

2=>two
1=>one
4=>four

Chapter Summary

This chapter covered the following:

- The built-in Python mapping type: dict





Python 3 Beginner

Chapter **04**

Container superpowers

release 1.0.0

Converting sequence types

➡ Tuple to list

```
list(('Hello', 'World'))
```

```
['Hello', 'World']
```

➡ List to tuple

```
tuple([0, 1, 2])
```

```
(0, 1, 2)
```

➡ List to set

```
set([0, 1, 0, 1, 2, 0, 1, 3])
```

```
{0, 1, 2, 3}
```

List comprehensions

➡ Create new lists from concise expressions

```
[x*x for x in range(10)]
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

```
books = ['1984', 'The Stranger', 'The Pragmatic Programmer']
prices = [9, 12.71, 22.1]
[(b, f'{p:.2f}') for b, p in zip(books, prices)]
```

```
[('1984', '9.00'),
 ('The Stranger', '12.71'),
 ('The Pragmatic Programmer', '22.10')]
```

List comprehensions

➡ Use a condition

```
l = [91, 39, 40, 76, 48, 57, 32, 88, 67, 44]  
[x for x in l if (x % 2) == 0]
```

```
[40, 76, 48, 32, 88, 44]
```

➡ Iterate over 2 variables

```
[(x, y) for x in (0, 1, 2) for y in (1, 2, 3)]
```

```
[(0, 1), (0, 2), (0, 3), (1, 1), (1, 2), (1, 3), (2, 1), (2, 2), (2, 3)]
```

```
[(x, y) for x in (0, 1, 2) for y in (1, 2, 3) if x != y]
```

```
[(0, 1), (0, 2), (0, 3), (1, 2), (1, 3), (2, 1), (2, 3)]
```

List comprehensions

➡ Use a condition

```
l = [91, 39, 40, 76, 48, 57, 32, 88, 67, 44]  
[x for x in l if (x % 2) == 0]
```

```
[40, 76, 48, 32, 88, 44]
```

➡ Iterate over 2 variables

```
[(x, y) for x in (0, 1, 2) for y in (1, 2, 3)]
```

```
[(0, 1), (0, 2), (0, 3), (1, 1), (1, 2), (1, 3), (2, 1), (2, 2), (2, 3)]
```

```
[(x, y) for x in (0, 1, 2) for y in (1, 2, 3) if x != y]
```

```
[(0, 1), (0, 2), (0, 3), (1, 2), (1, 3), (2, 1), (2, 3)]
```

More comprehensions

➡ Set comprehension

```
{(-1) ** x for x in range(1000)}
```

```
{-1, 1}
```

➡ Dictionary comprehension

```
s = 'Hello World!'
```

```
{c: s.count(c) for c in s}
```

```
{' ': 1, '!': 1, 'H': 1, 'W': 1, 'd': 1, 'e': 1, 'l': 3, 'o': 2, 'r': 1}
```

Starred expressions

→ Use a * to "unroll" an iterable object into a sequence type

```
[*range(10)]
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
(*range(10),)
```

```
(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
```

```
squares = [x*x for x in range(10)]  
[*enumerate(squares)]
```

```
[(0, 0), (1, 1), (2, 4), (3, 9), (4, 16), (5, 25), (6, 36), (7, 49),  
(8, 64), (9, 81)]
```

any & all

➡ any checks if *at least one* of the items is True (or truthful)

```
any([False, False, False, False])
```

False

```
any([False, False, True, False])
```

True

```
any([0, 1])
```

True

any & all

➡ all checks if *every* item is True (or truthful)

```
all([True, True, True, True])
```

True

```
all([True, True, True, False])
```

False

```
all([0, 1])
```

False

any & all

⇒ any and all become powerful with list comprehensions

```
l = [0.9385169882705057, 0.9536987969133713, 0.48928055608280896, ...  
any([x < 0.1 for x in l])
```

True

```
all([type(x) == float for x in l])
```

True

```
any([x == 0.0 for x in l])
```

False

Let's write some code!

```
s = 'Hello World!'\n\n{c: s.count(c) for c in s}
```

```
{' ': 1, '!': 1, 'H': 1, 'W': 1, 'd': 1, 'e': 1, 'l': 3, 'o': 2, 'r':\n1}
```

Workout Time!

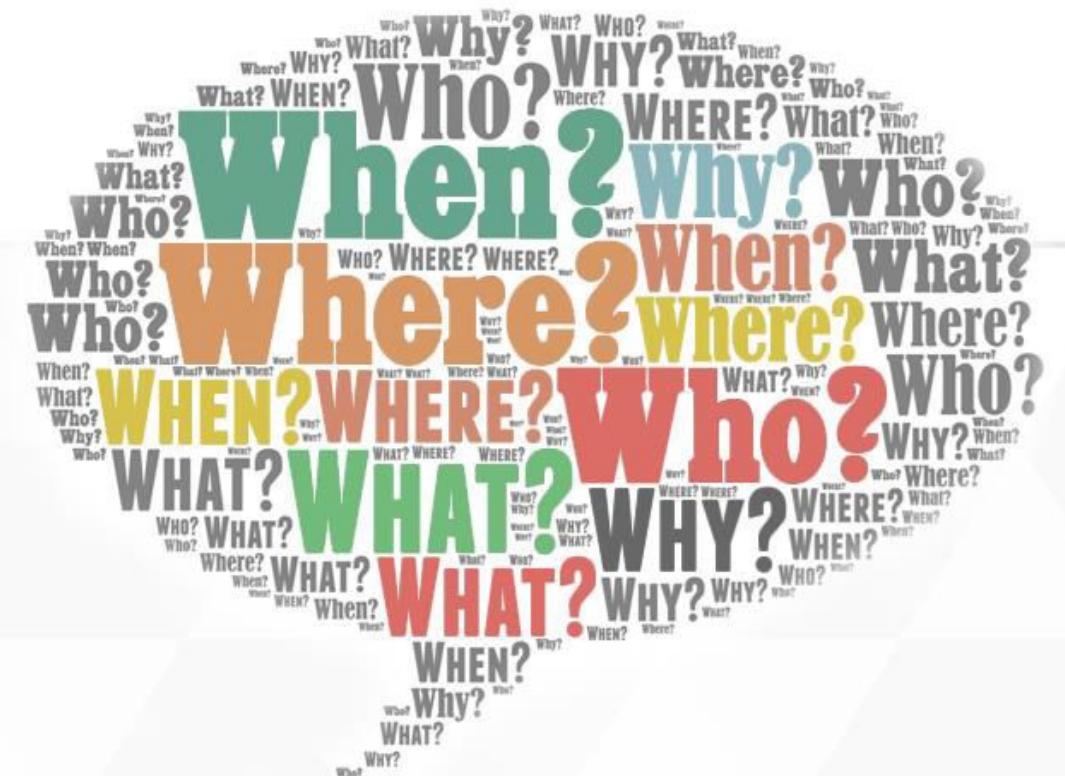


[Chapter 3-4 Exercises](#) link

Chapter Summary

This chapter covered the following:

- ➡ Converting sequence types between one another
- ➡ Using list comprehensions to create new lists in an expressive yet concise manner
- ➡ Unpacking iterable objects with starred expressions
- ➡ The `any` and `all` built-in functions and how to use them to maximum potential with list comprehensions



Q&A

Workshop



[Session 03 Workshop link](#)



**Thank you for
your attention!**