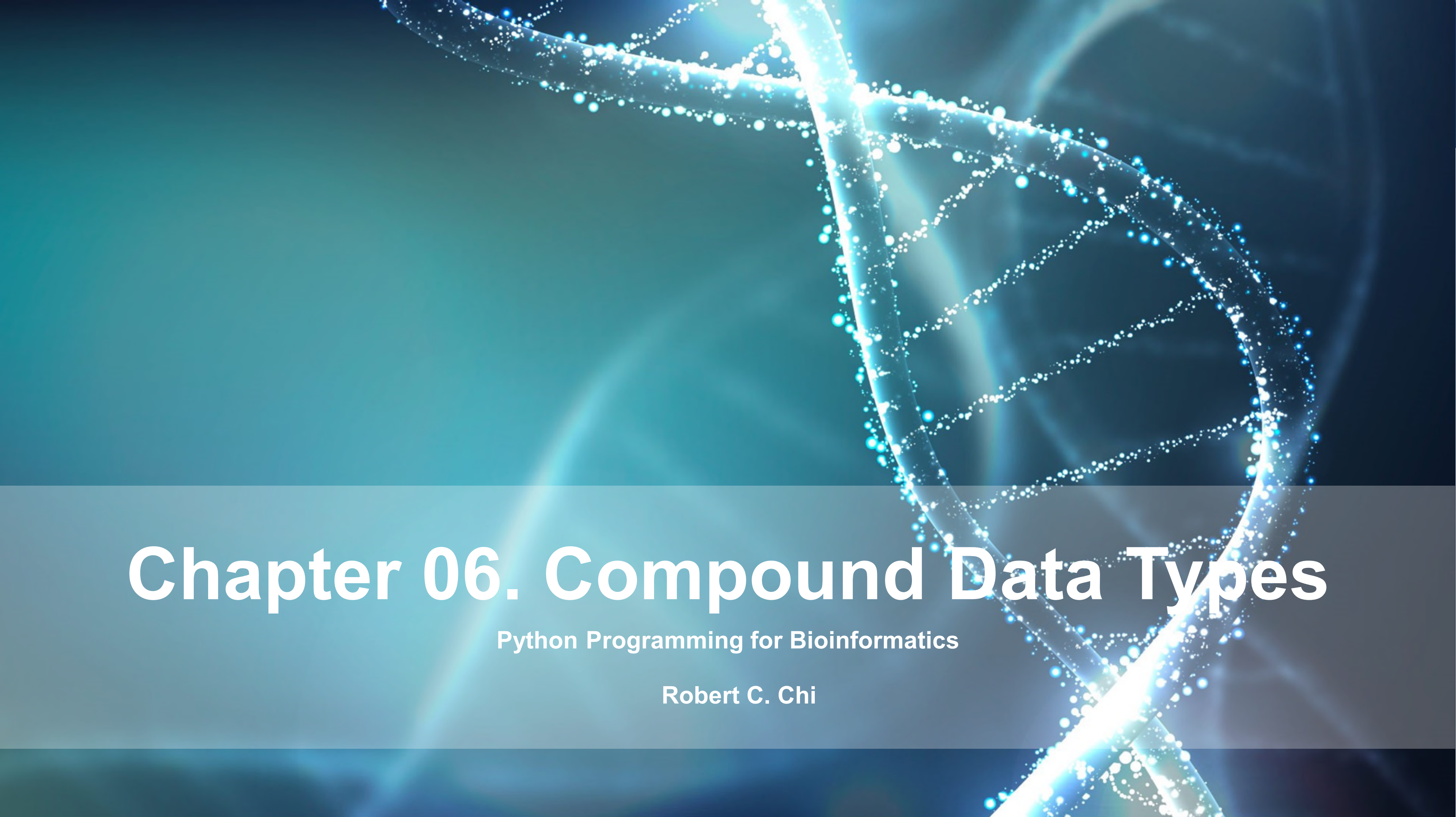


Chapter 06. Compound Data Types

Python Programming for Bioinformatics

Robert C. Chi

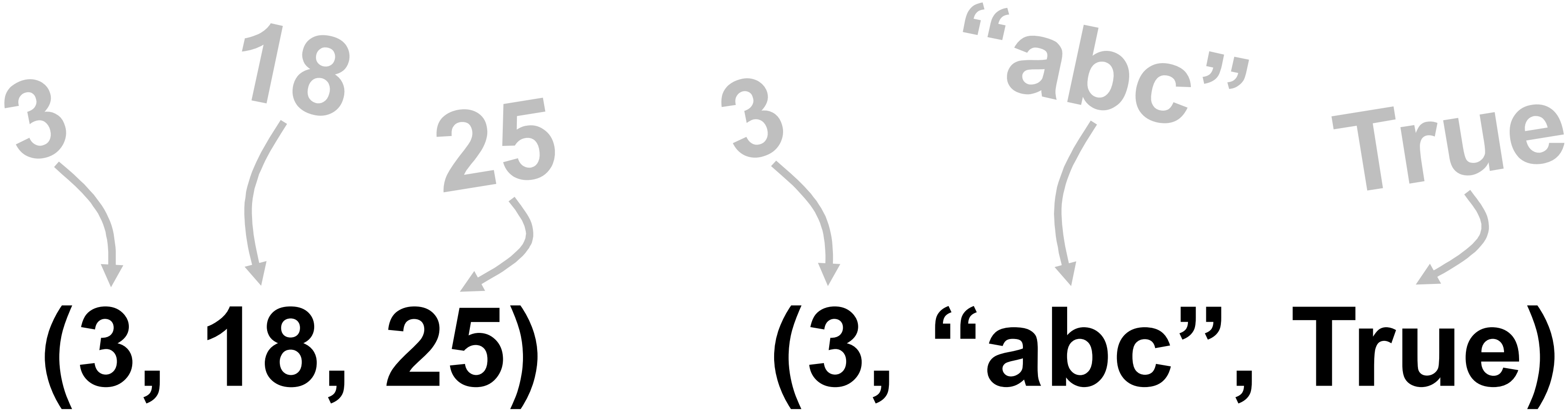
Agenda

- Introduction
- Tuple
- List
- Dictionary (Dict)
- Set

INTRODUCTION

What is "Compound Data Type"?

- The data type combining several "Literals"



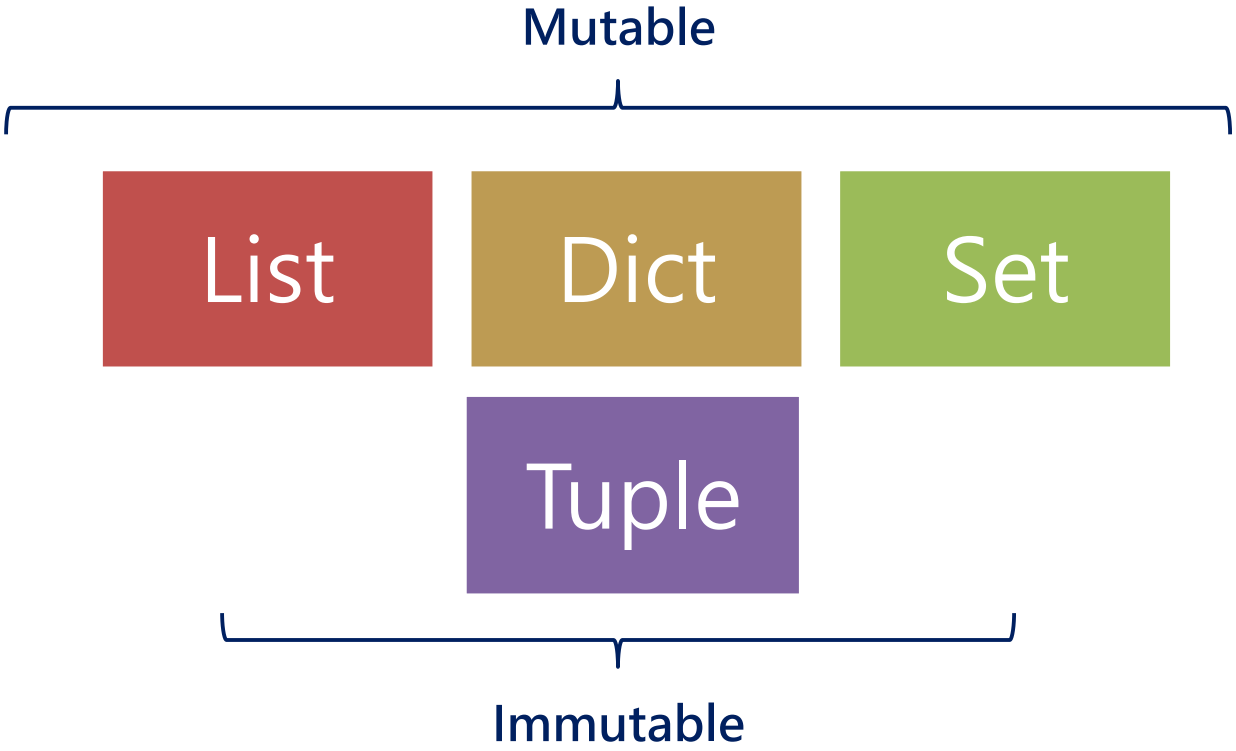
3 18 25

(3, 18, 25)

3 “abc” True

(3, “abc”, True)

Categories



Mutable

List

Dict

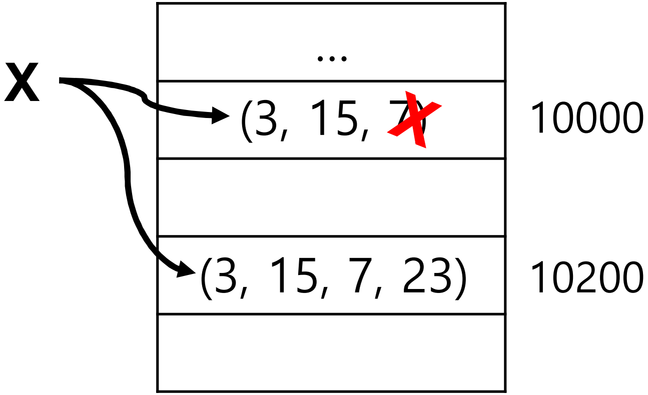
Set

Tuple

Immutable

Categories

- **What is "Immutable"?**
  - The old memory will be discarded when there is any modification.
  - i.e., “Once a memory was created, it will never be changed.”



X

...

(3, 15, 7)

10000

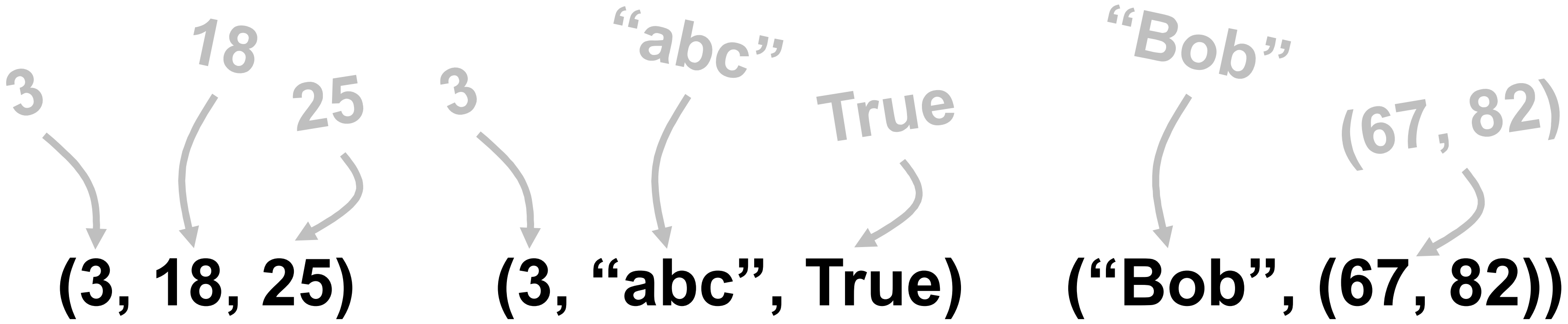
(3, 15, 7, 23)

10200

TUPLE

What is "Tuple"?

- A set of literals enclosed by ( ) and delimited by ,



(3, 18, 25)

(3, “abc”, True)

(“Bob”, (67, 82))

Tuple with Same Types of Literals

Tuple with Different Types of Literals

Tuple with Nested Tuples

Create a Tuple

- **Empty Tuples**
  - t = ()
  - t = tuple()

- **Tuples with Single Element**
  - t = “dog”, ← "comma" is mandatory, otherwise variable t will become a string
  - t = (“dog”,) ← "comma" is mandatory. You may check data type by type() command

- **Regular Tuples**
  - t = “dog”, “cat”
  - t = (“dog”, “cat”)

Un-packing & Exchange

- **Un-packing**
  - x, y, z = (3, 19, 23)
  - x → 3, y → 19, z → 23

- **Exchange of Values**
  - x = 3; y = 19
  - y, x = x, y
  - y → 3, x → 19

Concatenation & Repeats

- **Concatenation**
  - (2, 3) + (4, 5) → (2, 3, 4, 5)

- **Repeats**
  - (2, 3) * 3 → (2, 3, 2, 3, 2, 3)

Inclusion = in

2 in (2, 3, 4) → True

(2, 3) in (2, 3, 4) → False

Slicing

- t = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
- t[0] → 0
- t[2:7] → (2, 3, 4, 5, 6)
- t[-6:-3] → (4, 5, 6)
- t[2:] → (2, 3, 4, 5, 6, 7, 8, 9)
- t[:7] → (0, 1, 2, 3, 4, 5, 6)
- t[2:7:2] → (2, 4, 6)
- sc = slice(2, 7, 2)
  t[sc] → (2, 4, 6)

Length, Max & Min, Summation

- **Length**
  - len((1, 2, 3, 4, 5)) → 5

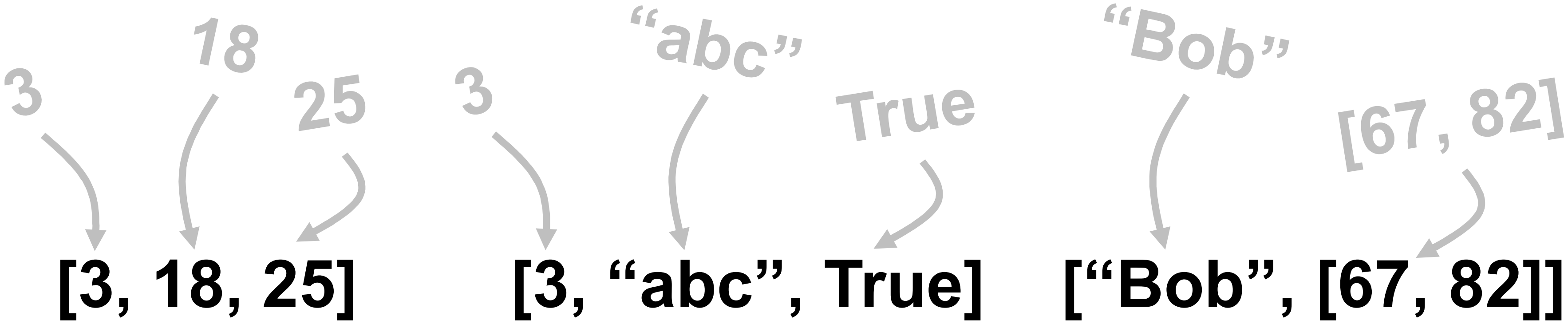
- **Maximum & Minimum**
  - min((1, 2, 3, 4, 5)) → 1
  - max((1, 2, 3, 4, 5)) → 5

- **Summation**
  - sum((1, 2, 3, 4, 5)) → 15

LIST

What is "List"?

- Similar to "Tuple" but surround elements by [ ].



3 18 25

[3, 18, 25]

List with Same Types of Literals

3 “abc” True

[3, “abc”, True]

List with Different Types of Literals

“Bob” [67, 82]

[“Bob”, [67, 82]]

List with Nested Lists or Tuples

Difference with Tuple

- **It’s Mutable**

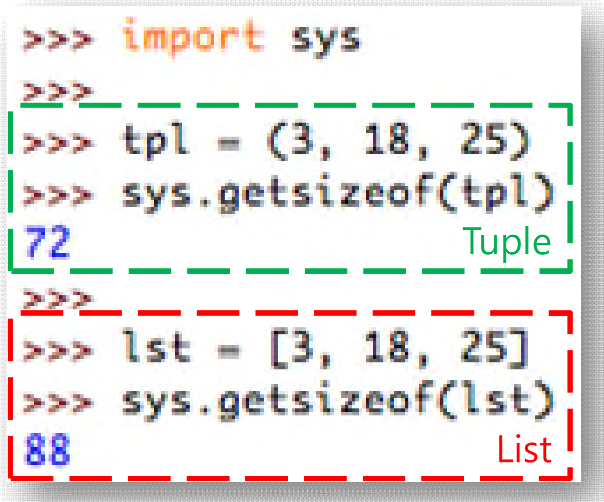
list1 = [3, 18, 25] +

- insert function
- modify function
- delete function

Provides a set of functions to change contents

> **Note** :
> - The original address could be changed or maintained after altering the contents. It depends on the system.

- **Larger Memory Allocation**



```
>>> import sys
>>>
>>> tpl = (3, 18, 25)
>>> sys.getsizeof(tpl)
72
>>>
>>> lst = [3, 18, 25]
>>> sys.getsizeof(lst)
88
```

Create a List

- **Empty Lists**
  - lst = []
  - lst = list()

- **Regular Lists**
  - lst = [3, 18, 25]

- **Lists with Various Data Types**
  - lst = [3, “abc”, True]

- **Nested Lists**
  - lst = [3, “abc”, [25, 6]]
  - lst = [3, “abc”, (25, 6)]

Un-packing, Concatenation, Repeats

- **Un-packing**
  - x, y, z = [1, 2, 3]
    → x = 1; y = 2; z = 3

- **Concatenation**
  - [1, 2] + [3, 4] → [1, 2, 3, 4]
  - lst = [1, 2]; lst.**append**([3, 4]) → [1, 2, [3, 4]]
  - lst = [1, 2]; lst.**extend**([3, 4]) → [1, 2, 3, 4]

- **Repeats**
  - lst = [1, 2, 3] * 3
    → lst = [1, 2, 3, 1, 2, 3, 1, 2, 3]

Inclusion = in

2 in [2, 3, 4] → True

[2, 3] in [2, 3, 4] → False

Length, Max & Min, Summation

- **Length**
  - len([1, 2, 3, 4, 5]) → 5

- **Max & Min**
  - min([1, 2, 3, 4, 5]) → 1
  - max([1, 2, 3, 4, 5]) → 5

- **Summation**
  - sum([1, 2, 3, 4, 5]) → 15

Reverse & Sort

- **Reverse**
  - list_iter = reversed([2, 32, 1, 6, 63, 9])
    list(list_iter) → [9, 63, 6, 1, 32, 2]
  - Return an Iterator from reversed()

- **Sort**
  - sorted([2, 32, 1, 6, 63, 9]) → [1, 2, 6, 9, 32, 63]
  - Return a "list" from sorted()

Find, Insert, Count

- **Find**
  - [1, 2, 3, 4, 5].**index**(3)
    → 2 # Found! Send the index back
  - [1, 2, 3, 4, 5].**index**(6)
    → ValueError # Not found! Send an error message back

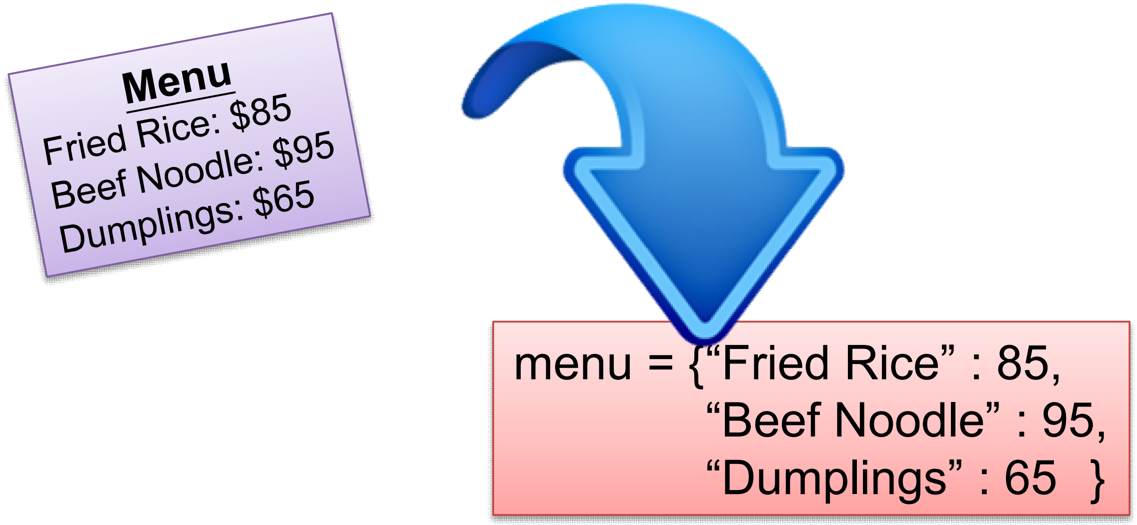
- **Insert**
  - [1, 2, 3, 4, 5].**insert**(2, 3) # 2: Index 3: Element
    → [1, 2, 3, 3, 4, 5]

- **Count**
  - [1, 2, 3, 3, 4, 5].**count**(3)
    → 2 # 3 appeared 2 times, returned 2

DICTIONARY (DICT)

What is "Dict"

- **The Data Structure that stores a "Mapping Table" and enclosed by { }**



Menu
Fried Rice: $85
Beef Noodle: $95
Dumplings: $65

```
menu = {“Fried Rice” : 85,
          “Beef Noodle” : 95,
          “Dumplings” : 65   }
```

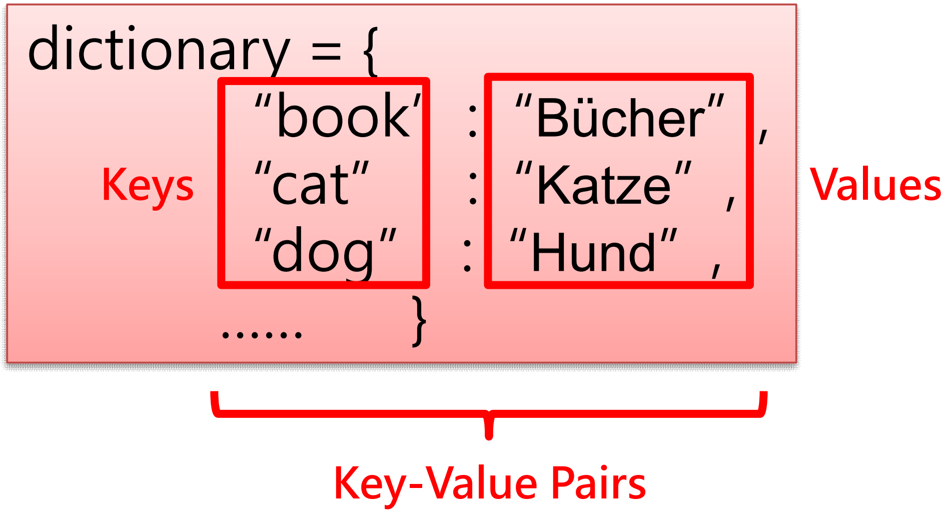
Why is it called "Dict"?

- **A "dictionary" is also a kind of "mapping table"**

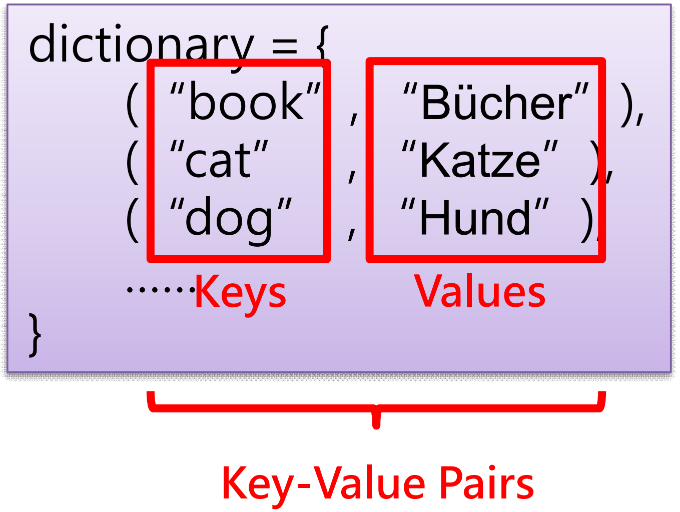
```
dictionary = {
            “book” : “Bücher”,
            “cat” : “Katze”,
            “dog” : “Hund”,
            ……      }
```

Terminologies of "Dict"

- **Keys, Values, Key-Value Pairs**
  - → In fact, Python uses tuple(Key, Value) to store a key-value pair.



```
dictionary = {
          “book’  : “Bücher” ,
          “cat”   : “Katze”  ,
          “dog”   : “Hund”   ,
          ......        }
```



```
dictionary = {
        ( “book” , “Bücher” ),
        ( “cat”  , “Katze”  ),
        ( “dog”  , “Hund”   ),
        ......
}
```

Applications of "Dict"

- **Lookup**

```
encryption = {
            “A”   :  “@”  ,
            “B”   :  “M”  ,
            “C”   :  “$”  ,
         ……     }
```

“ABC” ⇨ “@M$”

- **Translate**

```
exchanges = {
            “USD”   : 30.24,
            “JPY”   : 0.276,
            “RMB”   : 4.541,
         ……     }
```

Create a "Dict"

- **Empty Dict**
  - d = {}
  - d = **dict**()

- **Regular Dict**
  - Normal Way: d = {“USD”:30.24, “JPY”:0.276, “RMB”:4.541}
  - By zip(): d = **dict**(**zip**(("USD", "JPY", "RMB"), (30.24, 0.276, 4.541)))

Read and Modify

- **Read**
  - d = {“USD”:30.24, “JPY”:0.276, “RMB”:4.541}
  - d[“USD”] → 30.24
  - d[“EUR”] → Return KeyError when key doesn't exist

- **Modify**
  - d = {“USD”:30.24, “JPY”:0.276, “RMB”:4.541}
  - d[“USD”] = 31.02
    → d = {“USD”:**31.02**, “JPY”:0.276, “RMB”:4.541}
  - d[“EUR”] = 35.636 → Add a new element when it hasn't existed

Merge

- **Merge Two Dictionaries**
  - d1 = dict(zip(“abc”, range(1,4))) → {‘a’: 1, ‘b’: 2, ‘c’: 3}
  - d2 = dict(zip(“efg”, range(4, 7))) → {‘e’: 4, ‘f’: 5, ‘g’: 6}
  - d1.**update**(d2) → {‘a’: 1, ‘b’: 2, ‘c’: 3, ‘e’: 4, ‘f’: 5, ‘g’: 6}
  - If there is a duplicate key, the latter will override the former

Read Key & Value

- **Read Key**
  - d = dict(zip(“abc”, range(1, 4))) → {‘a’: 1, ‘b’: 2, ‘c’: 3}
  - list(d.**keys()**) → [‘a’, ‘b’, ‘c’]
  - tuple(d.**keys()**) → (‘a’, ‘b’, ‘c’)

- **Read Value**
  - d = dict(zip(“abc”, range(1, 4))) → {‘a’: 1, ‘b’: 2, ‘c’: 3}
  - list(d.**values()**) → [1, 2, 3]
  - tuple(d.**values()**) → (1, 2, 3)

- **Read (Key, Value)**
  - d = dict(zip(“abc”, range(1, 4))) → {‘a’: 1, ‘b’: 2, ‘c’: 3}
  - list(d.**items()**) → [(‘a’, 1), (‘b’, 2), (‘c’, 3)]
  - tuple(d.**items()**) → (('a', 1), ('b', 2), ('c', 3))

Delete

- **Delete an Element**
  - d = dict(zip(“abc”, range(1, 4))) → {‘a’: 1, ‘b’: 2, ‘c’: 3}
  - del d[“c”] → {‘a’: 1, ‘b’: 2}

- **Delete all Elements but Keep the Memory Allocation**
  - d = dict(zip(“abc”, range(1, 4))) → {‘a’: 1, ‘b’: 2, ‘c’: 3}
  - d.**clear**() → {}

- **Delete all Elements and Recycle the Memory Allocation**
  - d = dict(zip(“abc”, range(1, 4))) → {‘a’: 1, ‘b’: 2, ‘c’: 3}
  - del d

Length, Minimum, Maximum

- **Length**
  - d = dict(zip(“abc”, range(1, 4))) → {‘a’: 1, ‘b’: 2, ‘c’: 3}
  - **len**(d) → 3

- **Minimum**
  - d = dict(zip(“abc”, range(1, 4))) → {‘a’: 1, ‘b’: 2, ‘c’: 3}
  - **min**(d) → Find the minimum in “Key” → “a”
  - **min**(d.keys()) → Find the minimum in “Key” → “a”
  - **min**(d.values()) → Find the minimum in “Value” → 1

- **Maximum**
  - d = dict(zip(“abc”, range(1, 4))) → {‘a’: 1, ‘b’: 2, ‘c’: 3}
  - **max**(d) → Find the maximum in “Key” → “c”
  - **max**(d.keys()) → Find the maximum in “Key” → “c”
  - **max**(d.values()) → Find the maximum in “Value” → 3

Sorting

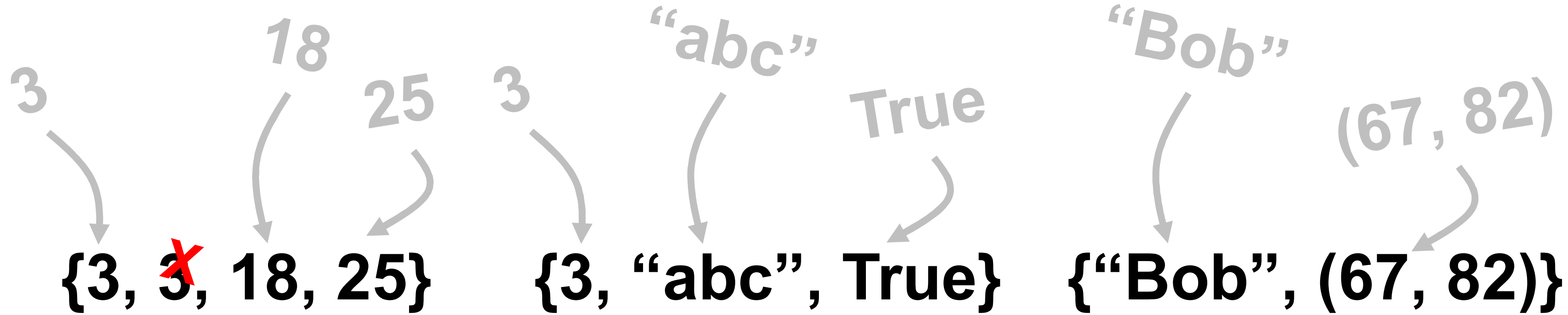
- **Sort by Keys**
  - d = {"b":2, "a":1, "c":3}
  - **sorted**(d) → ['a', 'b', 'c’]
  - **sorted**(d.**keys**()) → ['a', 'b', 'c’]

- **Sort by Values**
  - d = {"b":2, "a":1, "c":3}
  - **sorted**(d.**values**()) → [1, 2, 3]

SET

What is a "Set"?

- A Group of **Non-duplicated**, **Unordered** Data enclosed by **{ }**
  - → In fact, it is the part of "**Key**" in a "dictionary".



{3, ~~3~~, 18, 25}   {3, “abc”, True}   {“Bob”, (67, 82)}

Non-duplicated, Unordered Data

Available to Contain different types of data

Able to Contain Immutable Elements

Create a Set

- **Empty Set**
  - s = set() → set()
  - s = {} → X Treated as a dict
- **Regular Set**
  - s = {1, 2, 3, 4} → {1, 2, 3, 4}
  - s = {1, 2, 3, 3, 4} → {1, 2, 3, 4}
  - s = {3, “abc”, True} → {‘abc’, True, 3}
  - s = {“Bob”, (67, 82)} → {‘Bob’, (67, 82)}
  - s = {“Bob”, [67, 82]} → X The elements must be Immutable

Add, Include, Delete

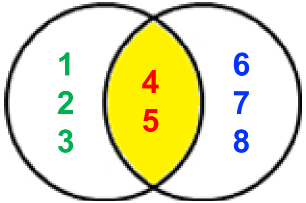
- **Add**
  - s = {1, 2, 3}
  - s.**add**(4) → {1, 2, 3, 4}

- **Include**
  - s = {1, 2, 3}
  - 2 **in** s → **True**

- **Delete**
  - Delete an Element
    - s = {1, 2, 3}
    - s.**remove**(3) → {1, 2}
  - Delete all Elements but Keep the Memory
    - s.**clear**() → set()
  - Delete both Elements and Memory
    - **del** s

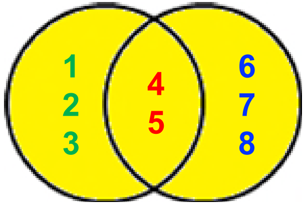
Operations of Sets

a = {1, 2, 3, 4, 5}
b = {4, 5, 6, 7, 8}



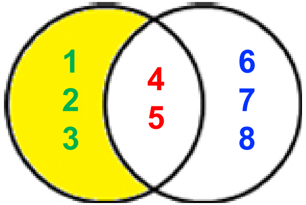
a & b = {4, 5}

Intersect

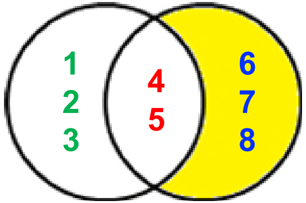


a | b =
{1, 2, 3, 4, 5, 6, 7, 8}

Union

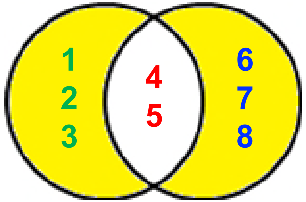


a - b = {1, 2, 3}



b - a = {6, 7, 8}

Complement



a ^ b = {1, 2, 3, 6, 7, 8}

Exclusive

Length, Minimum, Maximum

- **Length**
  - s = {1, 2, 3, 4, 5}
  - len(s) → 5

- **Minimum**
  - s = {1, 2, 3, 4, 5}
  - min(s) → 1

- **Maximum**
  - s = {1, 2, 3, 4, 5}
  - max(s) → 5

Summation, Sorting

- **Summation**
  - s = {1, 2, 3, 4, 5}
  - sum(s) → 15

- **Sorting**
  - s = {3, 1, 5, 4, 2}
  - sorted(s) → [1, 2, 3, 4, 5]
  - Note
    - sorted() return a “List” as the result.

Brief of this Section

- **The Data Structures You've Learned:**

Name	Immutable	Duplicable	Ordered	Applications
Tuple	v	v	v	Data that less Modified
List		v	v	Data that more Modified
Dict		Not for Keys		Data with Mapping Relationships
Set				Data that never duplicated