第四讲 字典、星号与模块 Lecture 4 Dictionary, Asterisk and Module

> 明玉瑞 Yurui Ming yrming@gmail.com

### 声明 Disclaimer

本讲义在准备过程中由于时间所限,所用材料来源并未规范标示引用来源。所引材料仅用于教学所用,作者无意侵犯原著者之知识版权,所引材料之知识版权均归原著者所有;若原著者介意之,请联系作者更正及删除。

The time limit during the preparation of these slides incurs the situation that not all the sources of the used materials (texts or images) are properly referenced or clearly manifested. However, all materials in these slides are solely for teaching and the author is with no intention to infringe the copyright bestowed on the original authors or manufacturers. All credits go to corresponding IP holders. Please address the author for any concern for remedy including deletion.

Python中的键值对是一对相互关联的值,键通过冒号关联到它的值。键的值可以是任何可以在 Python 中创建的对象,包括数字、字符串、列表及后面要讲的字典与类等。

A key-value pair is a set of values associated with each other; every key is connected to its value by a colon. A key's value can be any object that one can create in Python, such as a number, a string, a list, or another dictionary or class covered later.

▶ Python中的字典由键值对组成的集合构成,即包裹在大括号{}内的由逗号分隔包含一系列键值对构成。可以使用键来访问与该键关联的值。理论上,可以在字典中存储任意数量的键值对。

A dictionary in Python is a collection of key-value pairs, aka, a series of key-value pairs wrapped in braces, {}. One can use a key to access the value associated with that key. In theory dictionary can store as many key-value pairs.

- 通过在字典名称后跟一组方括号,方括号内写键的名称,即可获取字典中存储的与键关联的值。
  - To get the value associated with a key, give the name of the dictionary and then place the key inside a set of square brackets.
- ▶ Python中的字典是动态结构,即随时可以将新的键值对添加到字典中。要添加一个新的键值对,可以在字典名称后跟的方括号中给出新键,后跟新值。。
  - Dictionaries are dynamic structures, new key-value pairs could be added into a dictionary at any time. To add a new key-value pair, give the name of the dictionary followed by the new key in square brackets along with the new value.
- ▶ 从 Python 3.7 开始, 键值对在字典的存储顺序与它们添加到字典中的顺序相同。。
  - As of Python 3.7, dictionaries retain the order of key-values pairs in storage in accordance with the sequence that they were added to the dictionary.

▶ 有时从一个空字典开始,然后将每个新项逐渐添加到其中,会很方便。可以通过一对空的大括号定义一个空的字典。

It's sometimes convenient to start with an empty dictionary and then add each new item to it. An empty dictionary could be defined via an empty set of braces.

▶ 要修改字典中的值,可以在字典名称后跟的方括号中给出键名,然后是要与该键关联的新值。

To modify a value in a dictionary, give the name of the dictionary with the key in square brackets and then the new value to be associated with that key.

▶ 可以使用 del 语句删除存储在字典中的不再需要的键值对, del 需要后跟字典名称与位于中括号中的要删除的键。

del statement can be used to remove a no longer needed key-value pair stored in a dictionary. All del needs is the name of the dictionary and the key in the square brackets to be removed.

#### 字典

# Dictionary

▶ 使用方括号中写键的方式从字典中检索感兴趣的值可能会导致一个潜在的问题: 如果要求的键不存在,则将收到错误消息。针对这种情况,可以使用 get()方法设置一个默认值,如果请求的键不存在,将返回设定的值。get()方法第一个参数为指定查找的键,第二个可选参数设定为如果键不存在时,需要返回的值。

Using keys in square brackets to retrieve the interested value from a dictionary might cause one potential problem: if the key doesn't exist, it will trigger an error. For this instance, one can use the get() method to set a default value that will be returned if the requested key doesn't exist. The get() method requires a key as a first argument. As a second optional argument, it is the value to be returned if the key doesn't exist.

```
>>> student = {"Name": "Andy", "ID": "6729120", "Literature": 90, "Math": 92}
>>> print(student)
{'Name': 'Andy', 'ID': '6729120', 'Literature': 90, 'Math': 92}
>>> print(f" {student['Name']} got {student['Math']} for Math")
Andy got 92 for Math
>>> print(f" {student['Name']} got {student['History']} for History")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'History'
>>> print(f" {student['Name']} got {student.get('History', 60)} for History")
Andy got 60 for History
```

#### 字典

# Dictionary

▶ 如果在get()的调用中未提供第二个参数并且键不存在, Python将返回值None。特殊值 None 表示"不存在的值"。这不是错误:它是一个特殊值,表示没有值。

If the second argument in the call to get() are left out and the key doesn't exist, Python will return the value None. The special value None means "no value exists." This is not an error: it's a special value meant to indicate the absence of a value..

- ▶ 某些情况下可能需要遍历字典,这种情况下基本有3种形式:
  - ▶ 遍历字典中所有元素,
  - ▶ 遍历所有键,
  - ▶ 遍历所有值。

In some scenario we need to loop through the dictionary, there are basically three cases:

- Looping through all key-value pairs,
- ▶ Looping through all the keys in a dictionary,
- ▶ Looping through all values in a dictionary.

```
>>> student = {"Name": "Andy", "ID": "6729120", "Literature": 90, "Math": 92}
>>> for key, value in student.items():
... print("{}: {}".format(key, value))
Name: Andy
ID: 6729120
Literature: 90
Math: 92
>>> for key in student.keys():
          print(key)
Name
Literature
Math
>>> for value in student.values():
          print(value)
Andy
6729120
```

▶ 有时可能会希望以不同的顺序循环字典。一种方法是对在for循环中返回的键进行排序。可以使用 sorted() 函数按顺序获取键的副本。

Sometimes, one might need to loop through a dictionary in a different order. One way to do this is to sort the keys as they're returned in the for loop. one can use the sorted() function to get a copy of the keys in order.

```
// favorite_languages = {'jen': 'python', 'sarah': 'c', 'edward': 'ruby', 'phil': 'python', }
// for name, name_sorted in zip(favorite_languages.keys(), sorted(favorite_languages.keys())):
// print(f"{name.title()}/{name_sorted.title()}, thank you for taking the poll.")
// Jen/Edward, thank you for taking the poll.
// Sarah/Jen, thank you for taking the poll.
// Edward/Phil, thank you for taking the poll.
// Phil/Sarah, thank you for taking the poll.
// Phil/Sarah, thank you for taking the poll.
```

▶ 有时可能会希望循环字典时,输出的键或值不要重复。此时可以用集合。集合中的每个项目都必须是唯一的。

Sometimes it may be desirable to loop over a dictionary so that the output keys or values do not repeat. set can be used in this instance. Each item in the set must be unique.

```
>>> favorite_languages = {'jen': 'python', 'sarah': 'c', 'edward': 'ruby', 'phil': 'python',}
>>> print("The following languages have been mentioned:")
The following languages have been mentioned:
>>> for language in set(favorite_languages.values()):
... print(language.title())
...
Ruby
Python
C
>>>
```

▶ 读者可能会在很多地方看到 \* 和 \*\* 在 Python 中使用。 这两个运算符对于程序员新手和从许多其他可能没有完全等效运算符的编程语言迁移的人来说,有时可能有点神秘, 因此这里讨论一下这些运算符是什么以及它们的多种使用方式。\* 和 \*\* 运算符多年来的用途不断扩展,特别是Python 3 为这些运算符添加了许多新用途。

The readers might see there are a lot of places \* and \*\* used in Python. These two operators can be a bit mysterious at times, both for brand new programmers and for folks moving from many other programming languages which may not have completely equivalent operators. Here discusses what those operators are and the many ways they're used. The \* and \*\* operators have grown in ability over the years especially Python 3 has added a lot of new uses for these operators.

▶ 这里主要讨论 \* 和 \*\* 作为前缀运算符的情况,而不是中缀运算符的情况。 Here summarizes the cases \* and \*\* as prefix operators, not the infix operators.

▶ 下面第一例是 \* 和 \*\* 作为中缀运算符的情况,第二例是作为前缀运算符的情况。 The first example shows \* and \*\* as the prefix operators, while the second example is as infix operators.

```
>>> 2 * 5
10
>>> 2 ** 5
32
>>>
```

```
>>> numbers =list(range(5))
>>> more_numbers = [*numbers, 5, 6, 7, 8]
>>> print(*more_numbers, sep=', ')
0, 1, 2, 3, 4, 5, 6, 7, 8
>>>
```

▶ 星号可用于调用函数时将序列对象或可迭代对象,解包到函数调用的参数中。

Asterisks can be used to unpack a sequential object or iterable object into the arguments of a function call.

```
>>> fruits = ['lemon', 'pear', 'watermelon', 'tomato']
>>> print(fruits[0], fruits[1], fruits[2], fruits[3])
lemon pear watermelon tomato
>>> print(*fruits)
lemon pear watermelon tomato
```

▶ 如例中所示, print(\*fruits) 行将水果列表中的所有项目作为单独的参数传递给 print 函数调用, 我们甚至不需要知道列表中有多少个参数。

Just as in the example, that print(\*fruits) line is passing all of the items in the fruits list into the print function call as separate arguments, without us even needing to know how many arguments are in the list.

▶ \*运算符在不仅仅是语法糖。如果没有\*,这种将特定迭代中的所有项目作为单独 参数发送的能力是不可能的,除非列表是固定长度的。下面是一个展示这种更为 灵活性应用的例子。

The \* operator isn't just syntactic sugar. This ability of sending in all items in a particular iterable as separate arguments wouldn't be possible without \*, unless the list was a fixed length. The following is another example demonstrating this flexibility.

```
>>> def transpose_list(list_of_lists):
...     return [list(row) for row in zip(*list_of_lists)]
...
>>> transpose_list([[1, 4, 7], [2, 5, 8]])
[[1, 2], [4, 5], [7, 8]]
>>> transpose_list([[1, 4], [2, 5], [3, 6]])
[[1, 2, 3], [4, 5, 6]]
```

▶ \*\* 运算符做类似的事情,但用于关键字参数的情况。\*\*运算符允许我们根据键值 对字典将其解压缩到函数调用中的关键字参数中。

The \*\* operator does something similar, but with keyword arguments. The \*\* operator allows us to take a dictionary of key-value pairs and unpack it into keyword arguments in a function call.

```
>>> date_info = {'year': "2020", 'month': "01", 'day': "01"}
>>> filename = "{year}-{month}-{day}.txt".format(**date_info)
>>> print(filename)
2020-01-01.txt
```

▶ \*和\*\*都可以在函数调用中多次使用。

Both \* and \*\* can be used multiple times in function calls.

▶ 但是,用户在多次使用 \*\* 时需要小心。Python中的函数不能多次指定相同的关键字参数,因此与\*\*一起使用的每个字典中的键必须是不同的,否则将引发异常。

Users need to be careful when using \*\* multiple times though. Functions in Python can't have the same keyword argument specified multiple times, so the keys in each dictionary used with \*\* must be distinct or an exception will be raised.

```
>>> fruits = ['lemon', 'pear', 'watermelon', 'tomato']
>>> numbers = [2, 1, 3, 4, 7]
>>> print(*numbers, *fruits)
2 1 3 4 7 lemon pear watermelon tomato
```

```
>>> date_info = {'year': "2020", 'month': "01", 'day': "01"}
>>> track_info = {'artist': "Beethoven", 'title': 'Symphony No 5'}
>>> filename = "{year}-{month}-{day}-{artist}-{title}.txt".format(**date_info, **track_info)
>>> filename
'2020-01-01-Beethoven-Symphony No 5.txt'
```

```
def func(*, a, b):
    print(a)
    print(b)

func("gg") # TypeError: func() takes 0 positional arguments but 1 was given
func(a="gg") # TypeError: func() missing 1 required keyword-only argument: 'b'
func(a="aa", b="bb", c="cc") # TypeError: func() got an unexpected keyword argument 'c'
func(a="aa", b="bb", "cc") # SyntaxError: positional argument follows keyword argument
func(a="aa", b="bb") # aa, bb
```

▶ 正像之前讲的那样,定义函数时,\*运算符可用于捕获赋予函数的任意数量的位置参数。\*\*运算符可用于将给函数的任何关键字参数捕获到字典中。

Just as lectured before, when defining a function, the \* operator can be used to capture an unlimited number of positional arguments given to the function. The \*\* operator can be used to capture any keyword arguments given to the function into a dictionary.

▶ 有时,用为调用的明确性,特别会要求用关键字参数,则此时可以强制用单独的 \*符做到这一点。

keyword arguments sometimes are demanded for explicability, especially when default values are provided for multiple arguments, here bare \* is used to enforce this.

# 星号

#### Asterisk

▶ \*运算符也可以用于元组解包。

The \* operator can also be used in tuple unpacking.

▶ \* 也可以用于嵌套解包,但要注意该行为 正是读者正所期望的。

\* can be used for nested unpacking, but the reader should guarantee this is the desired behaviour.

```
>>> fruits = ['lemon', 'pear', 'watermelon', 'tomato']
>>> first, second, *remaining = fruits
>>> remaining
['watermelon', 'tomato']
>>> first, *remaining = fruits
>>> remaining
['pear', 'watermelon', 'tomato']
>>> first, *middle, last = fruits
>>> middle
['pear', 'watermelon']
```

```
>>> fruits = ['lemon', 'pear', 'watermelon', 'tomato']
>>> ((first_letter, *remaining), *other_fruits) = fruits
>>> remaining
['e', 'm', 'o', 'n']
>>> other_fruits
['pear', 'watermelon', 'tomato']
```

▶ \*运算符也可用于将不同类型的迭代连接在一起。 和 \*\*运算符也可用于将键/值 对从一个字典转储到新字典中。

\* operator can also be used to concatenate iterables of different types together. and \*\* operator can also be used for dumping key/value pairs from one dictionary into a new dictionary.

```
>>> fruits = ['lemon', 'pear', 'watermelon', 'tomato']
>>> numbers = list(range(5))
>>> mixed = [*fruits, *numbers]
>>> print(mixed)
['lemon', 'pear', 'watermelon', 'tomato', 0, 1, 2, 3, 4]
```

```
>>> date_info = {'year': "2020", 'month': "01", 'day': "01"}
>>> track_info = {'artist': "Beethoven", 'title': 'Symphony No 5'}
>>> all_info = {**date_info, **track_info}
>>> all_info
{'year': '2020', 'month': '01', 'day': '01', 'artist': 'Beethoven', 'title': 'Symphony No 5'}
```