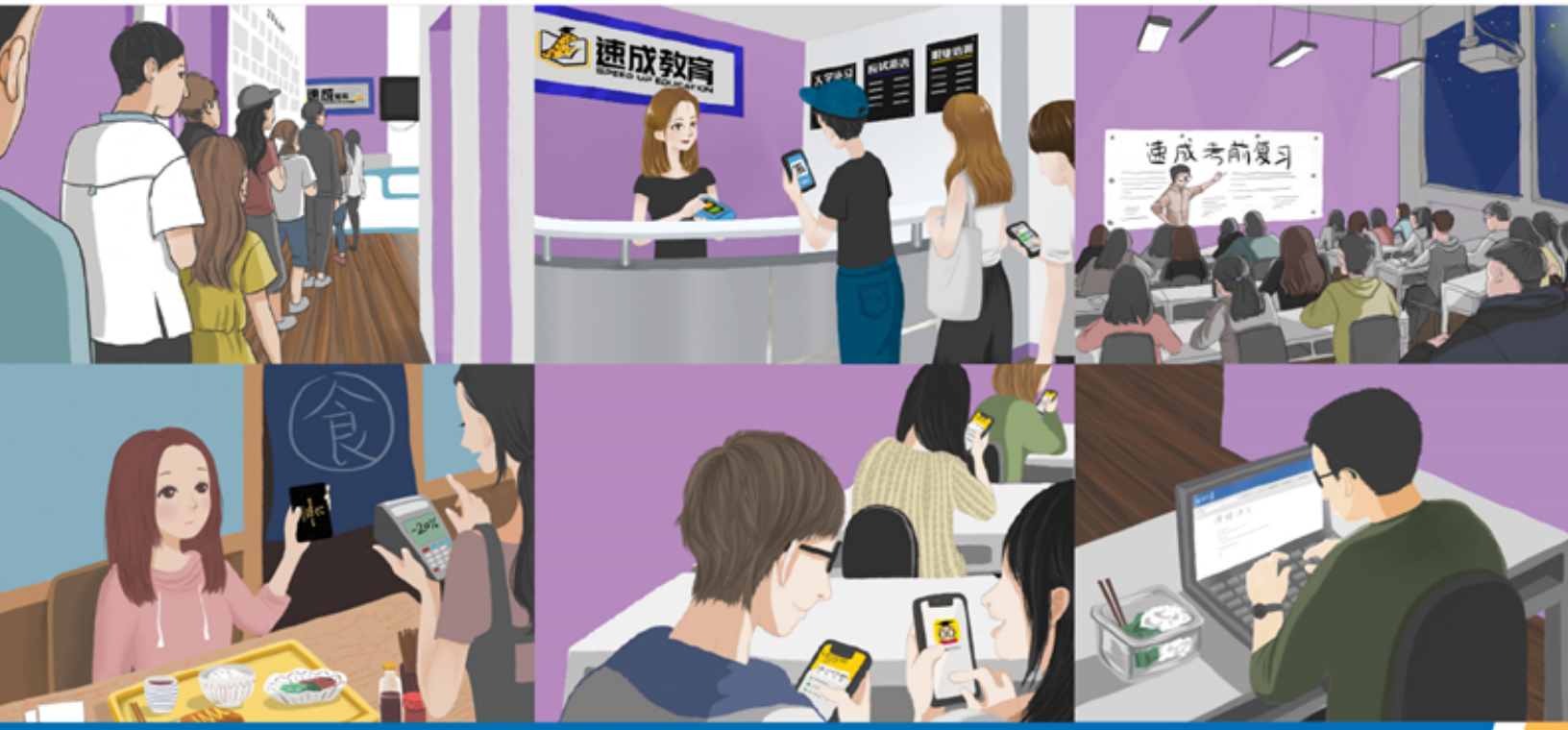
速成教育
SPEED UP EDUCATION



CSCA08

Introduction of Programming I

导师： **Vincent**

UTSC Dictionary | 2025/3/7

Disclaimer

This study handout is provided by Speed Up Education and its affiliated tutors. The study handout seeks to support your study process and should be used as a complement, not substitute to the university course material, lecture notes, problem, sets, past tests and other available resources.

The study handout is distributed for free to the students participating in Speed Up Education classes and is not for sale or other commercial uses whatsoever. We kindly ask you to refrain from copying or selling in part or in whole any information provided in the handout.

Thanks for choosing Speed Up Education. We wish you the best of luck in your exam.

Tuple

一种 collection, 类似 `list` 可以装很多 item, 用 `()` 表示

- 有顺序的, 所以和 `string`, `list` 一样, 也有 `index` 和 `slicing` 的功能
- 但是 `tuple` 是 `Immutable type` (不可被改变！！)
- 因此, `tuple` 没有任何 `method`
- 大多数时候, 可以不需要写 `()`, 两个 value 中间用 `,` 隔开 就默认是 `tuple`
- 如果要创建一个 长度为 1 的 `tuple` 必须加上一个`,` 在后面, Ex: `(3, )`

```
>>> names = ('Vincent', 'Alice', 'Bob')
>>> names[0]
'Vincent'
>>> names[2]
'Bob'

>>> names[0] = 'Anya'
TypeError: 'tuple' object does not support item assignment
```

通常会用 `tuple` 去实现 `parallel assignment` (同时赋值)

```
>>> x, y = 1, 2
>>> x
1
>>> y
2

>>> s = 'vincent,tse,18'
>>> first_name, last_name, age = s.split(',')

>>> first_name
'vincent'
>>> last_name
'tse'
>>> age
'18'
```

其他用法

```
data = [['vincent', 'tse'], ['anya', 'zhang'], ...]
for first_name, last_name in data:
    # do something ...
```

```
def func():
    a = ...
    b = ...
    return a, b

x, y = func()
```

Dictionary

一个 collection, 可以有很多个 `key: value` Pair, 每个 `key` 对应一个 `value`

`{key1: value1, key2: value2, key3: value3, ... }`

- `Dict` 是没有顺序的
- `key` 一定是 **Unique** (不会有重复) 并且必须是 `immutable type (int, float, str, tuple)`
- `value` 可以是任何 type

```
>>> name_to_gpa = {'Vc': 4.0, 'Alice': 2.7, 'Peter': 4.0, 'Bob': 0.7, 'Alice': 3.1}
>>> name_to_gpa
{'Vc': 4.0, 'Alice': 3.1, 'Peter': 4.0, 'Bob': 0.7}

>>> {['Alice', 'Chen']: 4.0, ['Anya', 'Cute']: 2.7}
TypeError: unhashable type: 'list'
```

常用的 Operation

`dict[key]`

- 得到 这个 `key` 在 `dict` 对应的 `value`,
- 如果 `key` 不在, 報錯: **KeyError**

```
>>> name_to_gpa = {'Vincent': 4.0, 'Alice': 2.7, 'Peter': 4.0, 'Bob': 0.7}
>>> name_to_gpa['Bob']
0.7

>>> name_to_gpa['Anya']
KeyError
```

`dict[key] = value`

- 如果 `key` 不在, 添加一个新的 `key: value`, 否则改变原本 `key` 的 `value`

```
>>> name_to_gpa = {'Vincent': 4.0, 'Alice': 2.7, 'Peter': 4.0, 'Bob': 0.7}
>>> name_to_gpa['Coral'] = 3.1
>>> name_to_gpa
{'Vincent': 4.0, 'Alice': 3.1, 'Peter': 4.0, 'Bob': 0.7, 'Carol': 3.1}

>>> name_to_gpa['Bob'] = 3.1
>>> name_to_gpa
{'Vincent': 4.0, 'Alice': 3.1, 'Peter': 4.0, 'Bob': 3.1, 'Carol': 3.1}
```

`del dict[key]`

- 把 `key` 和它对应的 `value` 从 `dict` 里删掉, 如果 `key` 不在, 報錯: **KeyError**

```
>>> name_to_gpa = {'Vincent': 4.0, 'Alice': 2.7, 'Peter': 4.0, 'Bob': 0.7}
>>> del name_to_gpa['Bob']
>>> name_to_gpa
{'Vincent': 4.0, 'Alice': 3.1, 'Peter': 4.0}
```

常用的 Methods

dict.keys()

- 返回一个 collection，内包含了全部的 key，顺序是从最先加进去的到最后加进去的
- 通常会把这个 collection 用 list() convert 成一个列表

```
>>> name_to_gpa = {'Vincent': 4.0, 'Alice': 2.7, 'Peter': 4.0, 'Bob': 0.7}
>>> names = list(name_to_gpa.keys())
>>> names
['Vincent', 'Alice', 'Peter', 'Bob']
```

dict.values()

- 返回一个 collection，内包含了全部的 value，顺序是从最先加进去的到最后加进去的
- 通常会把这个 collection 用 list() convert 成一个列表

```
>>> name_to_gpa = {'Vincent': 4.0, 'Alice': 2.7, 'Peter': 4.0, 'Bob': 0.7}
>>> names = list(name_to_gpa.values())
>>> names
[4.0, 2.7, 4.0, 0.7]
```

dict.update(dict2)

- 把 dict2 里全部的 key: value 都加进 dict 里 (直接 modify)
- 别忘了 key 是 Unique 的

```
>>> name_to_gpa = {'Vincent': 4.0, 'Alice': 2.7, 'Peter': 4.0, 'Bob': 0.7}
>>> name_to_gpa.update({'Alice': 4.0, 'Anya': 3.2}
>>> name_to_gpa
{'Vincent': 4.0, 'Alice': 4.0, 'Peter': 4.0, 'Bob': 0.7, 'Anya': 3.2}
```

Example: Find the name of the dumbest student ☺

```
def find_stupid(grades: dict[str, float]) -> str:
    name_lst = list(grades.keys())
    gpa_lst = list(grades.values())
    return name_lst[gpa_lst.index(min(gpa_lst))]


name_to_gpa = {'Vincent': 4.0, 'Alice': 2.7, 'Peter': 4.0, 'Bob': 0.7, 'Anya': 3.2}
stupid = find_stupid(name_to_gpa)
print('Stupid: ' + stupid)
```

Iterable Types (可以被一个一个看过去的):

Syntax

```
for <variable_name> in <collection>
```

```
for   char   in   'CSC108'
for   item   in   [1, 2, 3]
for   num    in   range(1, 9, 3)
for   item   in   (1, 2, 3)
for   key    in   {'a': 5, 'b': 4}
for   line   in   open("file", "r")
```

Example

```
name_to_gpa = {'vc': 4.0, 'alice': 3.2}
for name in name_to_gpa:
    gpa = name_to_gpa[name]
    print(name + ' got GPA: ' + str(gpa))
```

Output:

```
vc got GPA: 4.0
alice got GPA: 3.2
```

题型: Dictionary 的题目

```
def get_gpa_to_name(name_to_gpa: dict[str, float]) -> dict[float, list[str]]:
    """Return a new dictionary that maps each gpa to a list of names.

    >>> d = ({'alice': 4.0, 'bob': 2.7, 'carol': 4.0, 'david': 3.2}
    >>> d2 = get_gpa_to_name(d)
    >>> d2 == {4.0: ['alice', 'caorl'], 2.7: ['bob'], 3.2: ['david']}
    True
    """
```

```
def character_count(s: str) -> dict[str, int]:
    """Return a dictionary mapping each character in s and its number of
    occurrence.

    >>> character_count('i hate csca08')
    {'i': 1, 'h': 1, 'a': 2, 't': 1, 'e': 1, 'c': 2, 's': 1, '0': 1, '8': 1,
     ' ': 2}
    """
```

```
def get_membership(clubs_to_members: dict[str, list[str]])
        -> dict[str, list[str]]:
    """Return a dictionary whose keys are the members of all clubs in
    clubs_to_members, and whose values are the lists of clubs to which
    that member belongs.

    >>> clubs_to_members = {  'programming': ['Brian', 'Nick', 'Paco'],
    ...                       'games': ['Paco', 'Brian'], 'food': ['Paco'],
    ...                       'homework': []}

    >>> members_to_clubs = get_membership(clubs_to_members)
    >>> members_to_clubs == { 'Paco': ['food', 'games', 'programming'],
    ...                       'Brian': ['games', 'programming'],
    ...                       'Nick': ['programming']}
    True
    """
```

```
def get_positions(text: str) -> dict[str, list[int]]:
    """Return a dictionary where the keys are the individual words in
    text, and the values are the positions in the text where the words
    occur (starting at 1). Include punctuation and numbers in words,
    and convert alphabetic characters to lowercase.

    >>> result = get_positions('cats Cats CATS CATS!!!')
    >>> result == {'cats': [1, 2, 3], 'cats!!!': [4]}
    True
    >>> result = get_positions('I think I like CSCA08.')
    >>> result == {'i': [1, 3], 'think': [2], 'like': [4], 'cscA08.': [5]}
    True
    """
```

```
def can_fill_order(order_dict: dict[str, str],
                   inventory_dict: dict[str, int]) -> bool:
    """Return True iff the quantity (dict value) of every item (dict key) in
    inventory_dict is greater than or equal to the quantity of the item
    ordered in order_dict.

    If an item in order_dict is not in inventory_dict, return False.

    >>> inventory = {'shirt': 2, 'mug': 2}
    >>> can_fill_order({'Ann': 'mug', 'Bob': 'mug', 'Lee': 'mug'}, inventory)
    False
    >>> can_fill_order({'Ann': 'shirt', 'Bob': 'mug', 'Lee': 'mug'}, inventory)
    True
    >>> can_fill_order({'Ann': 'mug', 'Bob': 'mug', 'Lee': 'hat'}, inventory)
    False
    >>> inventory = {'shirt': 2, 'mug': 2}
    >>> order = {'Ann': 'mug', 'Bob': 'mug'}
    >>> result = can_fill_order(order, inventory)
    >>> order == {'Ann': 'mug', 'Bob': 'mug'}
    True
    >>> inventory == {'shirt': 2, 'mug': 2}
    True
    """
```

```
def find_population(continent_info: dict[str, dict[str, dict[str, int]]])
        -> dict[str,int]:
    """Return a dictionary where the keys are continents from continent_info
    and the values are the total population of all cities on that continent.

    >>> data = {'North America': {
    ...             'Canada': {'Toronto': 5000, 'Ottawa': 200},
    ...             'USA': {'Portland': 400, 'New York': 5000, 'Chicago': 1000},
    ...             'Mexico': {'Mexico City': 10000}},
    ...         'Asia': {
    ...             'Thailand': {'Bangkok': 456},
    ...             'Japan': {'Tokyo': 10000, 'Osaka': 5000}},
    ...         'Antarctica': {}}

    >>> result = find_population(data)
    >>> result == {'North America': 21600, 'Asia': 15456, 'Antarctica': 0}
    True
    """
```

```
def organize_walks(walks: list[tuple[str, str, int]]) -> dict[str, dict[str, int]]:
    """Return a dictionary based on the data in walks where the keys are dog names;
    the values are dictionaries where the keys are dog walker names and the values
    are the total distance that walker walked a particular dog.

    walks is a list of tuples, where each tuple has the form
    (<dog name>, <walker name>, <distance walked>) representing a single walk.

    >>> organize_walks([('Uli', 'Jeff', 3), ('Uli', 'Jeff', 5)])
    {'Uli': {'Jeff': 8}}
    >>> organize_walks([  ('Felix', 'Bob', 5), ('Fido', 'Bob', 2),
    ...                   ('Felix', 'Bob', 3), ('Fluffy', 'Ann', 10),
    ...                   ('Felix', 'Ann', 1), ('Fluffy', 'Ann', 10)])
    {'Felix': {'Bob': 8, 'Ann': 1}, 'Fido': {'Bob': 2}, 'Fluffy': {'Ann': 20}}
    """
```

Example

```
def mystery(d1: Dict[int, List[str]], d2: Dict[int, List[str]]) -> None
```

```
'''



Preconditions:









'''
```

```
for k in d2:
    for item in d2[k]:
        if item not in d1[k]:
            d1[k].append(item)
```