ListDict - Supplymentary

Ting ZHANG

2022.11

OUTLINE

- **Dictionary continued**
- **Nesting**
- **How does Python manage memory?**

Tranverse the dict

--- keys() 、 values()、items()

- keys() returns a sequence of all the keys in a dictionary
- values() returns a sequence of all the values in a dictionary
- items() returns a sequence of all the pairs in a dictionary

Similar to list **but not real list** **can not be changed**

```
spam = {'name': 'Polka', 'color': 'red', 'age': 42}
```

```
for v in spam.values():
    print(v)
```

```
Polka
red
42
```

```
for k in spam.keys():
    print(k)
```

```
name
color
age
```

```
for i in spam.items():
    print(i)
```

```
('name', 'Polka')
('color', 'red')
('age', 42)
```

```
spam = {'name': 'Polka', 'color': 'red', 'age': 42}
```

```
for i in spam.items():
    print(i)
```

```
('name', 'Polka')
('color', 'red')
('age', 42)
```

```
for k, v in spam.items():
    print(k + ' = ' + str(v))
```

```
name = Polka
color = red
age = 42
```

Set (remove repetition)

- set_variable = {item1, item2, ···, itemN}

```
programming_languages = {'python', 'ruby', 'python', 'c'}
print(programming_languages)
```

```
{'ruby', 'python', 'c'}
```

Summary

- list_variable = [item1, item2, …, itemN]
- tuple_variable = (item1, item2, …, itemN) **fixed**
- dictionary_variable = {pair_item1, pair_ item2, …, pair_ itemN}
- set_variable = {item1, item2, …, itemN} **remove repetition**

Nesting

- List of dictionaries
- Store list in dictionary
- Store dictionary in dictionary

List of dictionaries

```
pet_0 = {'name': 'Polka', 'species': 'dog', 'age': 5}
pet_1 = {'name': 'Sophie', 'species': 'cat', 'age': 8}
pet_2 = {'name': 'Jerry', 'species': 'rabbit', 'age': 3}
pets = [pet_0, pet_1, pet_2]
print(f'Total number of pets: {len(pets)}')
for pet in pets:
    print(pet)
```

```
Total number of pets: 3
{'name': 'Polka', 'species': 'dog', 'age': 5}
{'name': 'Sophie', 'species': 'cat', 'age': 8}
{'name': 'Jerry', 'species': 'rabbit', 'age': 3}
```

Store list in dictionary

```
pet = {'name': 'Sophie', 'species': 'cat', 'age': 8, 'favorite_foods': ['fish', 'chicken', 'mouse']}
print(f"{pet['name']} is a {pet['species']}. Her/His favorite_foods are: ")
for food in pet['favorite_foods']:
    print(food)
```

```
Sophie is a cat. Her/His favorite_foods are:
fish
chicken
mouse
```

Store dictionary in dictionary

```
users = {
    'aeinstein': {
        'first': 'albert',
        'last': 'einstein',
        'location': 'princeton',
        },
    'mcurie': {
        'first': 'marie',
        'last': 'curie',
        'location': 'paris',
        },
    }
```

```python
for username, user_info in users.items():
    print(f"Username: {username}")
    full_name = f"{user_info['first']} {user_info['last']}"
    location = user_info['location']
    print(f"\tFull name: {full_name.title()}")
    print(f"\tLocation: {location.title()}")
```

```
Username: aeinstein
    Full name: Albert Einstein
    Location: Princeton
Username: mcurie
    Full name: Marie Curie
    Location: Paris
```

How does Python manage memory?

- list: can be changed

- tuple and str: fixed, can not be changed

```
dimensions = (200, 50)
print(dimensions[0])
dimensions[0] = 250
dimensions = (400, 100)
```

√

√

X Tuples don't support item assignment

√

How does Python manage memory?

- list: can be changed

- tuple and str: fixed, can not be changed

```
user_name = 'admin'
print(user_name[0])
user_name[5] = '1'
user_name = 'admin1'
```

√

√

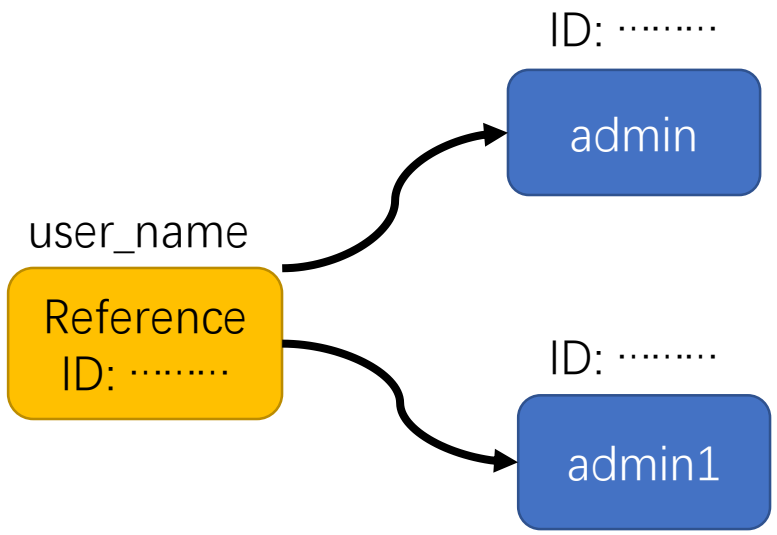
X

```
    user_name[5] = '1'
TypeError: 'str' object does not support item assignment
```

√

In Python, each value possess only one identifier.

```
user_name = 'admin'
print(id(user_name))
user_name = 'admin1'
print(id(user_name))
```

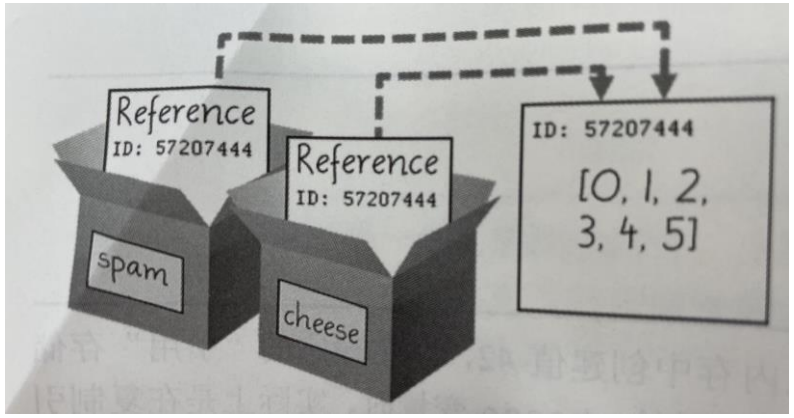


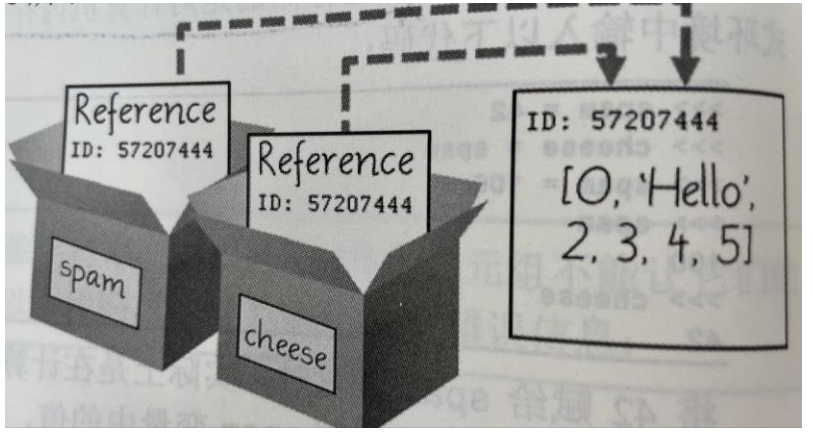

```
2609570977080
2609570977136
```

Variable stores the reference of memory block in Python (For C\ C++, it is different)

An example for List

```
spam = [0, 1, 2, 3, 4, 5]
cheese = spam
cheese[1] = 'Hello!'
print(spam)
print(cheese)
```






```
[0, 'Hello!', 2, 3, 4, 5]
[0, 'Hello!', 2, 3, 4, 5]
```

How does Python manage memory?

- Reference counting mechanism

  Python internally uses reference counting (recording how many references an object has) to keep track of objects in memory.

- Garbage collection

  The garbage collector reclaims objects with a reference count of 0

- Memory pool mechanism

  A memory pool mechanism to manage the application and release of small memory.