



Trường ĐH Khoa Học Tự Nhiên Tp. Hồ Chí Minh

TRUNG TÂM TIN HỌC



Lập trình Python cơ bản

Bài 7: List – Tuple – Dictionary – Set



Phòng Lập Trình - Mạng

2023

Nội dung

1. List

2. Tuple

3. Dictionary

4. Set



1. List

- ❑ Là cấu trúc dữ liệu cơ bản trong Python, thuộc nhóm sequence.
- ❑ Một list bao gồm nhiều phần tử, mỗi phần tử trong list có một vị trí gọi là index. Phần tử đầu tiên có **index = 0**, phần tử cuối cùng có **index = số phần tử - 1**.
- ❑ List là tập hợp có thứ tự các phần tử (*elements*) thuộc nhiều kiểu dữ liệu khác nhau (*như strings, integers, objects, other lists, ...*). Tuy nhiên, nếu các phần tử có cùng kiểu dữ liệu thì dễ xử lý, tính toán hơn.

1. List

- Cú pháp: Sử dụng cặp dấu ngoặc vuông [] để khai báo một list.

[phần_tử_1, phần_tử_2,... , phần_tử_n]

- Ví dụ:

- Khai báo list rỗng:

```
list_1 = []
```

- Khai báo và gán giá trị:

```
list_2 = [1, 3, 5, 7, 9]
```

```
list_3 = ['Xuân', 'Hạ', 'Thu', 'Đông']
```

```
list_4 = [10, 9.75, ['Lập trình', 'Mạng']]
```

- Khai báo và gán giá trị dựa vào hàm: range(start, stop, step)

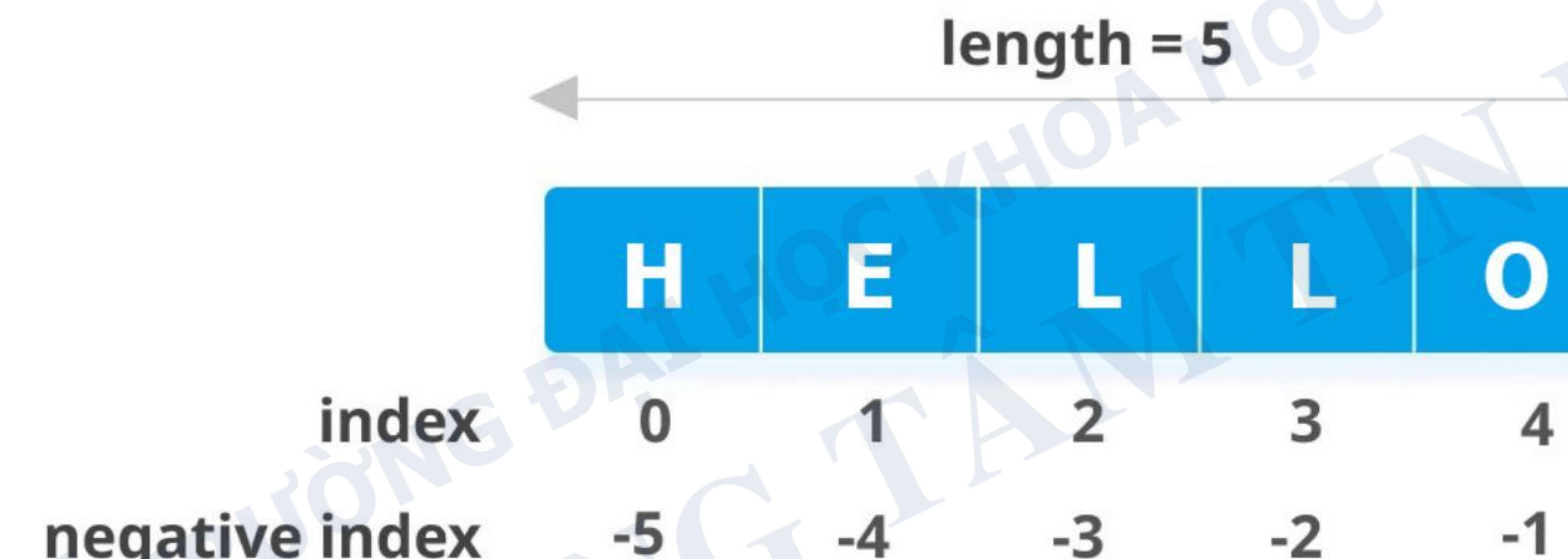
```
list_5 = range(1, 11)
```

Kết quả:

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

1. List

□ Truy xuất phần tử trong list



1. List

□ Truy xuất phần tử trong list

- Truy xuất index:

ten_list[index]

```
my_list = ['p', 'y', 't', 'h', 'o', 'n']
print(my_list[1])                      # y
print(my_list[len(my_list)-1])          # n
```

- Truy xuất negative index: **ten_list[index]**

```
my_list = ['p', 'y', 't', 'h', 'o', 'n']
print(my_list[-1])                     # n
print(my_list[-len(my_list)])           # p
```

1. List

□ Truy xuất phần tử trong list

- Tạo list con từ list gốc:

ten_list[index_dau : index_cuo]

```
list_s = ['H', 'A', 'P', 'P', 'Y', ' ', 'B', 'I', 'R', 'T', 'H', 'D', 'A', 'Y']

# Lấy 5 phần tử đầu tiên
sub_list_1 = list_s[:5]
print(sub_list_1)          # ['H', 'A', 'P', 'P', 'Y']

# Lấy 3 phần tử cuối cùng
sub_list_2 = list_s[-3:]
print(sub_list_2)          # ['D', 'A', 'Y']

# Lấy các phần tử từ index 3 đến cận 7
sub_list_3 = list_s[3:7]
print(sub_list_3)          # ['P', 'Y', ' ', 'B']
```

1. List

□ Cập nhật phần tử trong list

```
list_numbers = [1, 3, 5, 7, 9]
```

- Cập nhật 1 phần tử: **ten_list[index] = gia_tri**

```
list_numbers[0] = 100  
print(list_numbers) # [100, 3, 5, 7, 9]
```

- Cập nhật danh sách các phần tử liên tục:

```
list_numbers[1:4] = [20, 40, 60]  
print(list_numbers) # [100, 20, 40, 60, 9]
```

1. List

□ Các toán tử cơ bản

- Toán tử +:

list_moi = list_1 + list_2 + ... + list_n

```
list_1 = [1, 2]
list_2 = [3, 4, 5]
list_3 = [6, 7, 8, 9]
list_4 = list_1 + list_2 + list_3
print(list_4) # [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

1. List

❑ Các toán tử cơ bản

- Toán tử *: `list_moi = ten_list * n` (n: số lần)

```
list_1 = [1, 2]
print(list_1 * 3) # [1, 2, 1, 2, 1, 2]
```

- Toán tử thành phần: `gia_tri in ten_list`

```
list_3 = [6, 7, 8, 9]
print(7 in list_3) # True
print('8' in list_3) # False
```

1. List

□ Các phương thức cơ bản

- Số phần tử trong list: **len(ten_list)**

```
my_list = [1, 2, 3, 4, 5, 6, 7, 8, 9]
print(len(my_list)) # 9
```

- Giá trị lớn nhất, nhỏ nhất: **max(ten_list) / min(ten_list)**

```
my_list = [1, 2, 3, 4, 5, 6, 7, 8, 9]
print('Giá trị lớn nhất:', max(my_list))
print('Giá trị nhỏ nhất:', min(my_list))
```

Giá trị lớn nhất: 9
Giá trị nhỏ nhất: 1

1. List

❑ Các phương thức cơ bản

- Duyệt list: **for item in ten_list**

```
danh_sach_so = [1, 2, 3, 4, 5, 6, 7, 8, 9]
tong = 0
for so in danh_sach_so:
    tong += so
else:
    print('Tổng =', tong)
```

Tổng = 45

1. List

❑ Các phương thức cơ bản

- Tính tổng: **sum(ten_list)**

```
danh_sach_so = [1, 2, 3, 4, 5, 6, 7, 8, 9]
print('Tổng =', sum(danh_sach_so))
print('Giá trị trung bình:', sum(danh_sach_so) / len(danh_sach_so))
```

Tổng = 45

Giá trị trung bình: 5.0

1. List

□ Các phương thức cơ bản

- Thêm phần tử vào cuối list: **ten_list.append(phan_tu)**

```
list_animals = []
print('List ban đầu:', list_animals)
list_animals.append('dog')
list_animals.append('cat')
list_animals.append('monkey')
list_animals.append('duck')
list_animals.append('elephant')
print('List sau khi thêm:', list_animals)
```

List ban đầu: []

List sau khi thêm: ['dog', 'cat', 'monkey', 'duck', 'elephant']

1. List

□ Các phương thức cơ bản

- Thêm phần tử vào list tại vị trí index: **ten_list.insert(index, phan_tu)**

```
list_animals = ['dog', 'cat', 'monkey', 'duck', 'elephant']
list_animals.insert(1, 'bird')
print(list_animals)
```

```
['dog', 'bird', 'cat', 'monkey', 'duck', 'elephant']
```

1. List

❑ Các phương thức cơ bản

- Mở rộng list: **ten_list_muon_mo_rong.extend(ten_list)**

```
list_1 = [1, 2, 3]
list_2 = [4, 5, 6]
list_1.extend(list_2)
print(list_1)
```

```
[1, 2, 3, 4, 5, 6]
```

1. List

□ Các phương thức cơ bản

- Tạo bản sao từ list cho trước: **ten_list.copy()**

```
list_chars = ['p', 'y', 't', 'h', 'o', 'n']
print('list_chars:', list_chars)
list_chars_copy = list_chars.copy()
print('list_chars_copy:', list_chars_copy)
```

```
list_chars: ['p', 'y', 't', 'h', 'o', 'n']
list_chars_copy: ['p', 'y', 't', 'h', 'o', 'n']
```

1. List

❑ Các phương thức cơ bản

- Đếm số lần xuất hiện của một phần tử trong list: **ten_list.count(phan_tu)**

```
list_animals = ['dog', 'cat', 'monkey', 'dog', 'duck', 'elephant']
print(list_animals.count('dog')) # 2
print(list_animals.count('cat')) # 1
```

1. List

□ Các phương thức cơ bản

- Tìm index của element trong list: **ten_list.index(phan_tu)**

- Trả về index nhỏ nhất nếu tìm thấy
- Nếu không tìm thấy sẽ thông báo lỗi

```
list_animals = ['dog', 'cat', 'monkey', 'dog', 'duck', 'elephant']
print(list_animals.index('dog')) # 0
```

```
print(list_animals.index('bird'))
```

```
Traceback (most recent call last):
  File "k:/GIANG_DAY/_PORTAL/Chuyen_de
    print(list_animals.index('bird'))
ValueError: 'bird' is not in list
```

1. List

❑ Các phương thức cơ bản

- Lấy 1 phần tử ra khỏi list: **ten_list.pop([index])**

```
list_animals = ['dog', 'cat', 'monkey', 'dog', 'duck', 'elephant']
```

```
list_animals.pop()  
print(list_animals)
```

```
['dog', 'cat', 'monkey', 'dog', 'duck']
```

```
list_animals.pop(3)  
print(list_animals)
```

```
['dog', 'cat', 'monkey', 'duck']
```

1. List

❑ Các phương thức cơ bản

- Xóa phần tử trong list: **del(ten_list[index])**

```
list_animals = ['dog', 'cat', 'monkey', 'duck', 'elephant']
del(list_animals[3])
print(list_animals)
```

```
['dog', 'cat', 'monkey', 'elephant']
```

- Xóa phần tử trong list: **ten_list.remove(phan_tu)**

```
list_animals = ['dog', 'cat', 'monkey', 'duck', 'elephant']
list_animals.remove('monkey')
print(list_animals)
```

```
['dog', 'cat', 'duck', 'elephant']
```

1. List

❑ Các phương thức cơ bản

- Xóa tất cả các phần tử trong list: **ten_list.clear()**

```
list_animals = ['dog', 'cat', 'monkey', 'duck', 'elephant']
list_animals.clear()
print(list_animals)
```

```
[]
```

1. List

❑ Các phương thức cơ bản

- Đảo ngược list: **ten_list.reverse()**

```
list_numbers = [1, 8, 2, 3, 7, 10, 4]
print('Before: ', list_numbers)
list_numbers.reverse()
print('After: ', list_numbers)
```

Before: [1, 8, 2, 3, 7, 10, 4]
After: [4, 10, 7, 3, 2, 8, 1]

1. List

□ Các phương thức cơ bản

- Sắp xếp list:

```
list_numbers = [1, 8, 2, 3, 7, 10, 4]
```

- Tăng dần: **ten_list.sort()**

```
print('Before:', list_numbers)
list_numbers.sort()
print('After: ', list_numbers)
```

Before: [1, 8, 2, 3, 7, 10, 4]
After: [1, 2, 3, 4, 7, 8, 10]

- Giảm dần: **ten_list.sort(reverse=True)**

```
print('Before:', list_numbers)
list_numbers.sort(reverse=True)
print('After: ', list_numbers)
```

Before: [1, 8, 2, 3, 7, 10, 4]
After: [10, 8, 7, 4, 3, 2, 1]

1. List

- ❑ **List Comprehension:** cho phép dùng cú pháp lệnh đơn giản để tạo ra list mới từ list có sẵn
- ❑ Cú pháp 1: `new_list = [expression for item in iterable]`

```
# In hoa các phần tử trong list
list_animals = ['dog', 'cat', 'monkey', 'duck', 'elephant']
list_animals_upper = [animal.upper() for animal in list_animals]
print(list_animals_upper)

['DOG', 'CAT', 'MONKEY', 'DUCK', 'ELEPHANT']
```

1. List

- ❑ **List Comprehension:** cho phép dùng cú pháp lệnh đơn giản để tạo ra list mới từ list có sẵn
- ❑ Cú pháp 2: **new_list = [expression for item in iterable if condition]**

```
# Lấy ra tên những con thú có 3 ký tự
list_animals = ['dog', 'cat', 'monkey', 'duck', 'elephant']
list_animals_new = [animal for animal in list_animals if len(animal) == 3]
print(list_animals_new)
```

```
[ 'dog', 'cat' ]
```

1. List

- ❑ **List Comprehension:** cho phép dùng cú pháp lệnh đơn giản để tạo ra list mới từ list có sẵn
- ❑ Cú pháp 3: **new_list = [expr1 if condition else expr2 for item in iterable]**

```
# Tạo list kết quả (Đậu/Rớt) dựa vào list điểm có sẵn  
# Kết quả Đậu nếu điểm >= 5, ngược lại Rớt  
list_diem = [8, 9.25, 4, 7, 10, 3, 6]  
list_ket_qua = ['Đậu' if diem >= 5 else 'Rớt' for diem in list_diem]  
print(list_ket_qua)
```

```
[ 'Đậu', 'Đậu', 'Rớt', 'Đậu', 'Đậu', 'Rớt', 'Đậu' ]
```

Nội dung

1. List

2. Tuple

3. Dictionary

4. Set



2. Tuple

- ❑ Là cấu trúc dữ liệu cơ bản trong Python, thuộc nhóm sequence.
- ❑ Một tuple bao gồm nhiều phần tử, mỗi phần tử trong tuple có một vị trí gọi là index. Phần tử đầu tiên có **index = 0**, phần tử cuối cùng có **index = số phần tử - 1**.
- ❑ Giới hạn, cố định số phần tử, không thay đổi giá trị phần tử.

2. Tuple

- Cú pháp: (**phần_tử_1, phần_tử_2, ..., phần_tử_n**)

- Có thể không sử dụng () khi tạo tuple

```
tuple_1 = (1, 8, 2, 3, 7, 10, 4)
```

```
print(tuple_1)
```

```
print(type(tuple_1))
```

```
(1, 8, 2, 3, 7, 10, 4)
```

```
<class 'tuple'>
```

```
tuple_2 = 'one', 'two', 'three', 'four', 'five'
```

```
print(tuple_2)
```

```
print(type(tuple_2))
```

```
('one', 'two', 'three', 'four', 'five')
```

```
<class 'tuple'>
```

2. Tuple

□ Cú pháp: (**phần_tử_1, phần_tử_2,... , phần_tử_n**)

- Có thể không sử dụng () khi tạo tuple
- Lưu ý: Khi tạo tuple có 1 phần tử, cần thêm dấu phẩy (,) vào sau phần tử đó.

```
tuple_3 = ('rose',)  
print(tuple_3)  
print(type(tuple_3))
```

```
('rose',)  
<class 'tuple'>
```

2. Tuple

□ Truy xuất phần tử trong tuple

- Truy xuất index:

ten_tuple[index]

```
tuple_2 = 'one', 'two', 'three', 'four', 'five'  
print(tuple_2[2]) # three
```

- Truy xuất negative index: **ten_tuple[index]**

```
tuple_2 = 'one', 'two', 'three', 'four', 'five'  
print(tuple_2[-2]) # four
```

2. Tuple

□ Truy xuất phần tử trong tuple

- Tạo tuple con từ tuple gốc: **ten_tuple[index_dau : index_cuo]**

```
tuple_1 = (1, 8, 2, 3, 7, 10, 4)
```

```
# Các phần tử từ index 2 đến cận 5
```

```
tuple_4 = tuple_1[2:5]
```

```
print(tuple_4) # (2, 3, 7)
```

```
# 3 phần tử đầu tiên
```

```
tuple_5 = tuple_1[:3]
```

```
print(tuple_5) # (1, 8, 2)
```

```
# 4 phần tử cuối cùng
```

```
tuple_6 = tuple_1[-4:]
```

```
print(tuple_6) # (3, 7, 10, 4)
```

2. Tuple

□ Cập nhật / xóa phần tử trong tuple

- **Lưu ý:** Do tuple không thay đổi, do đó ta **không thể** cập nhật / xóa phần tử trong tuple.
- Chỉ có thể xóa bỏ đối tượng (biến) tuple:

```
tuple_4 = (2, 3, 7)
del(tuple_4)
print(tuple_4)
```

Traceback (most recent call last):
 File "k:/GIANG_DAY/_PORTAL/Chuyen_de/LDS
 print(tuple_4)
NameError: name 'tuple_4' is not defined

2. Tuple

❑ Các phương thức cơ bản của tuple

- Cũng tương tự như các phương thức cơ bản của list nhưng **không có** phương thức: **sort**, **reverse**, **remove**, **pop**, **insert**, **extend**, **append**



Nội dung

1. List
2. Tuple
3. Dictionary
4. Set



3. Dictionary

- ❑ Là cấu trúc dữ liệu cơ bản trong Python, thuộc nhóm sequence.
- ❑ Một dictionary bao gồm nhiều phần tử, mỗi phần tử trong dictionary sẽ là một bộ **key : value** (cấu trúc này tương tự như 1 object json)
 - Key: là giá trị duy nhất (không trùng) trong dictionary và không thể chỉnh sửa. Kiểu dữ liệu của key có thể là **number, string, tuple**.
 - Value: có thể trùng và có thể cập nhật.

3. Dictionary

- Cú pháp: Sử dụng cặp dấu ngoặc nhọn {} để khai báo một dictionary.

{key_1: value_1, key_2: value_2, ..., key_n: value_n}

- Tạo dictionary:

- Khai báo dictionary rỗng: dict_1 = {}
print(dict_1) {}

- Khai báo và gán giá trị: dict_2 = {'x': 1, 'y': 2, 'z': 3}
print(dict_2)
print(type(dict_2))

```
{'x': 1, 'y': 2, 'z': 3}  
<class 'dict'>
```

3. Dictionary

- Cú pháp: Sử dụng cặp dấu ngoặc nhọn {} để khai báo một dictionary.

{key_1: value_1, key_2: value_2, ..., key_n: value_n}

- Tạo dictionary:

- Sử dụng hàm **dict()**:

```
dict_3 = dict([(1, 'Một'), (2, 'Hai')])
print('dict_3:', dict_3)
# hoặc
dict_4 = dict(zip([1, 2], ['Một', 'Hai']))
print('dict_4:', dict_4)

dict_3: {1: 'Một', 2: 'Hai'}
dict_4: {1: 'Một', 2: 'Hai'}
```

3. Dictionary

□ Truy xuất giá trị trong dictionary

- Cú pháp: `ten_dictionary[key]`

```
dict_mua = {1: 'Xuân', 2: 'Hạ', 3: 'Thu', 4: 'Đông'}  
print(dict_mua[1])      # Xuân
```

```
dict_so = {'one': 1, 'two': 2, 'three': 3, 'four': 4, 'five': 5}  
print(dict_so['three']) # 3
```

3. Dictionary

□ Cập nhật / thêm mới giá trị vào dictionary

- Cú pháp: `ten_dictionary[key] = gia_tri`
 - Thêm mới: với key chưa có trong dictionary.
 - Cập nhật: với key đã có trong dictionary.

```
# Cập nhật
dict_animals = {1: 'dog', 2: 'cat', 3: 'elephant', 4: 'buffalo', 5: 'bird'}
dict_animals[3] = 'lion'
print(dict_animals)

{1: 'dog', 2: 'cat', 3: 'lion', 4: 'buffalo', 5: 'bird'}
```



```
# Thêm mới
dict_animals = {1: 'dog', 2: 'cat', 3: 'elephant', 4: 'buffalo', 5: 'bird'}
dict_animals[6] = 'frog'
print(dict_animals)

{1: 'dog', 2: 'cat', 3: 'elephant', 4: 'buffalo', 5: 'bird', 6: 'frog'}
```

3. Dictionary

❑ Các phương thức cơ bản

- Tạo dictionary với danh sách các key từ sequence: `dict.fromkeys(seq[, value])`

```
list_nums = [1, 2, 3, 4, 5]
```

```
dict_1 = dict.fromkeys(list_nums)
print('dict_1:', dict_1)
dict_1: {1: None, 2: None, 3: None, 4: None, 5: None}
```

```
dict_2 = dict.fromkeys(list_nums, 'python')
print('dict_2:', dict_2)
dict_2: {1: 'python', 2: 'python', 3: 'python', 4: 'python', 5: 'python'}
```

3. Dictionary

□ Các phương thức cơ bản

- `len(ten_dictionary)`: Trả về số phần tử trong dictionary

```
dict_languages = {1: 'python', 2: 'php', 3: '.net', 4: 'java', 5: 'javascript'}  
print(len(dict_languages)) # 5
```

- `ten_dictionary.get(key)`: Trả về value dựa trên key

```
dict_languages = {1: 'python', 2: 'php', 3: '.net', 4: 'java', 5: 'javascript'}  
print(dict_languages.get(3))  
# tương đương  
print(dict_languages[3])
```

.net

3. Dictionary

❑ Các phương thức cơ bản

- `ten_dictionary.items()`: Trả về danh sách các bộ tuple (key, value) của dictionary

```
dict_languages = {1: 'python', 2: 'php', 3: '.net', 4: 'java', 5: 'javascript'}  
print(dict_languages.items())  
  
dict_items([(1, 'python'), (2, 'php'), (3, '.net'), (4, 'java'), (5, 'javascript')])
```

```
# Sử dụng vòng lặp  
for key, value in dict_languages.items():  
    print(key, '--', value)  
  
1 -- python  
2 -- php  
3 -- .net  
4 -- java  
5 -- javascript
```

3. Dictionary

□ Các phương thức cơ bản

- `ten_dictionary.keys()`: Trả về danh sách các **key** của dictionary

```
dict_languages = {1: 'python', 2: 'php', 3: '.net', 4: 'java', 5: 'javascript'}  
print(dict_languages.keys())  
  
dict_keys([1, 2, 3, 4, 5])
```

```
# Sử dụng vòng lặp  
for key in dict_languages.keys():  
    print(key)
```

1
2
3
4
5

3. Dictionary

❑ Các phương thức cơ bản

- **ten_dictionary.values()**: Trả về danh sách các **key** của dictionary

```
dict_languages = {1: 'python', 2: 'php', 3: '.net', 4: 'java', 5: 'javascript'}  
print(dict_languages.values())
```

```
dict_values(['python', 'php', '.net', 'java', 'javascript'])
```

```
# Sử dụng vòng lặp  
for value in dict_languages.values():  
    print(value)
```

```
python  
php  
.net  
java  
javascript
```

3. Dictionary

❑ Các phương thức cơ bản

- `ten_dictionary_1.update(ten_dictionary_2)`: Cập nhật các phần tử từ `ten_dictionary_2` vào `ten_dictionary_1`

```
dict_languages = {1: 'python', 2: 'php', 3: '.net', 4: 'java', 5: 'javascript'}
dict_languages_new = {6: 'ruby', 7: 'html', 8: 'perl'}
dict_languages.update(dict_languages_new)
print(dict_languages)
```

```
{1: 'python', 2: 'php', 3: '.net', 4: 'java', 5: 'javascript', 6: 'ruby', 7: 'html', 8: 'perl'}
```

3. Dictionary

□ Dictionary Comprehension

- **Dictionary Comprehension** được sử dụng khi đầu vào ở dạng **dictionary** hoặc cặp **key: value**
- Cú pháp 1: **new_dictionary = { key: value for (key,value) in iterable }**

```
days = [  
    "Monday", "Tuesday", "Wednesday",  
    "Thursday", "Friday", "Saturday", "Sunday"  
]  
temp_c = [30.5, 32.6, 31.8, 33.4, 29.8, 30.2, 29.9]  
weekly_temp = {day:temp for (day,temp) in zip(days, temp_c)}  
print(weekly_temp)
```

```
{'Monday': 30.5, 'Tuesday': 32.6, 'Wednesday': 31.8, 'Thursday': 33.4, 'Friday': 29.8, 'Saturday': 30.2, 'Sunday': 29.9}
```

3. Dictionary

□ Dictionary Comprehension

- **Dictionary Comprehension** được sử dụng khi đầu vào ở dạng **dictionary** hoặc cặp **key: value**
- **Cú pháp 2:** `new_dictionary = { key: value for (key, value) in iterable if condition }`

```
# Tạo dict lưu trữ các học sinh đạt điểm trên trung bình (>= 5)
danh_sach_hoc_sinh = ["Mai", "Nam", "Lan", "Trúc", "Minh", "Thái"]
danh_sach_diem = [9, 8.5, 3, 6, 7, 4]
d = zip(danh_sach_hoc_sinh, danh_sach_diem)
dict_ket_qua = {hoc_sinh: diem for (hoc_sinh, diem) in d if diem >= 5}
print(dict_ket_qua)

{'Mai': 9, 'Nam': 8.5, 'Trúc': 6, 'Minh': 7}
```

3. Dictionary

□ Dictionary Comprehension

- **Dictionary Comprehension** được sử dụng khi đầu vào ở dạng **dictionary** hoặc cặp **key: value**
- **Cú pháp 3:**

```
new_dictionary = { key: (value_if if condition else value_else) for (key, value) in iterable }
```

```
# Tạo dict lưu trữ các học sinh đạt điểm trên trung bình (>= 5)
danh_sach_hoc_sinh = ["Mai", "Nam", "Lan", "Trúc", "Minh", "Thái"]
danh_sach_diem = [9, 8.5, 3, 6, 7, 4]
d = zip(danh_sach_hoc_sinh, danh_sach_diem)
dict_ket_qua = {hoc_sinh: ('Đậu' if diem >= 5 else 'Rớt') for (hoc_sinh, diem) in d}
print(dict_ket_qua)
```

```
{'Mai': 'Đậu', 'Nam': 'Đậu', 'Lan': 'Rớt', 'Trúc': 'Đậu', 'Minh': 'Đậu', 'Thái': 'Rớt'}
```

Nội dung

1. List

2. Tuple

3. Dictionary

4. Set



4. Set

- ❑ Là cấu trúc dữ liệu cơ bản trong Python, thuộc nhóm sequence.
- ❑ Set gồm tập hợp các phần tử **độc nhất** (không trùng lặp) và có thể thực hiện các phép toán về tập hợp như: hợp, giao, ...
- ❑ Đặc điểm của set:
 - Không giới hạn số lượng phần tử.
 - Có thể thêm hoặc xóa phần tử.
 - Có thể chứa biến thuộc nhiều kiểu dữ liệu khác nhau, nhưng không thể chứa phần tử có thể thay đổi được như list, set hay dictionary.
 - Các phần tử trong set không theo thứ tự thêm vào, không sử dụng index.

4. Set

- Set được tạo bằng cách đặt tất cả các phần tử trong dấu ngoặc nhọn {}, phân tách nhau bằng dấu phẩy: { set1, set_2, ..., set_n }

```
set_nums = {1, 5, 4, 6, 3, 7}  
print(set_nums)  
print(type(set_nums))  
  
{1, 3, 4, 5, 6, 7}  
<class 'set'>
```

```
set_strings = {'Kim', 'Mộc', 'Thủy', 'Hỏa', 'Thổ'}  
print(set_strings)  
print(type(set_strings))  
  
{'Thổ', 'Mộc', 'Hỏa', 'Thủy', 'Kim'}  
<class 'set'>
```

- Nếu khởi tạo set ban đầu chưa có phần tử (set rỗng), sử dụng cú pháp: ten_set = set()

```
set_fruits = set()  
print(set_fruits)  
  
set()
```

4. Set

□ Thêm phần tử vào set

- **add()**: Thêm 1 phần tử vào set.

```
set_fruits = set()  
print(set_fruits)  
set_fruits.add('lemon')  
set_fruits.add('orange')  
set_fruits.add('apple')  
print(set_fruits)
```

```
set()  
{'orange', 'lemon', 'apple'}
```

- **update()**: Thêm nhiều phần tử vào set trong cùng 1 lần. **update()** có thể nhận *tuple*, *list*, *string* và *set* làm đối số. Trong mọi trường hợp, set có giá trị duy nhất, các bản sao sẽ tự động bị loại bỏ.

```
my_set = {4}  
my_set.update([1, 2, 3], 'Python')  
print(my_set)
```



```
{1, 2, 3, 4, 'n', 'P', 'y', 'o', 't', 'h'}
```

4. Set

❑ Kiểm tra phần tử có tồn tại trong set hay không

- Sử dụng toán tử thành phần **in** hoặc **not in**

```
my_set = {1, 2, 3, 4}  
print(3 in my_set)      # True  
print('o' in my_set)    # False
```

4. Set

❑ Xóa phần tử trong set

- Sử dụng hàm **discard()** hoặc **remove()**
- Cú pháp: `ten_set.discard('gia_tri_can_xoa')`
`ten_set.remove('gia_tri_can_xoa')`
- Khi phần tử cần xóa không tồn tại trong set thì **discard()** không làm gì cả, còn **remove()** sẽ báo lỗi.

```
set_strings = {'Kim', 'Mộc', 'Thủy', 'Hỏa', 'Thổ'}
set_strings.discard('Kim')
print(set_strings)          # {'Thủy', 'Thổ', 'Mộc', 'Hỏa'}
set_strings.remove('Mộc')
print(set_strings)          # {'Thủy', 'Thổ', 'Hỏa'}
set_strings.remove('Mộc')
print(set_strings)          # KeyError: 'Mộc'
```

4. Set

□ Xóa tất cả các phần tử trong set

- Cú pháp: `ten_set.clear()`

```
set_strings = {'Kim', 'Mộc', 'Thủy', 'Hỏa', 'Thổ'}
set_strings.clear()
print(set_strings)    # set()
```

□ Xóa set ra khỏi bộ nhớ

- Cú pháp: `del(ten_set)`

```
set_animals = {'bear', 'dog', 'lion', 'elephant', 'horse'}
del(set_animals)
print(set_animals)
```

Traceback (most recent call last):

```
  File "k:/GIANG_DAY/_PORTAL/Chuyen_de/LDS1_2
      print(set_animals)
NameError: name 'set_animals' is not defined
```

4. Set

□ Lấy phần tử ra khỏi set

- Cú pháp: `ten_set.pop()`
- Kết quả trả về là một phần tử trong set, báo lỗi nếu set không có phần tử cần lấy.

```
set_fruits = {'orange', 'lemon', 'apple', 'strawberry'}
fruit = set_fruits.pop()
print(fruit)      # strawberry
print(set_fruits) # {'orange', 'apple', 'lemon'}
```

4. Set

□ Duyệt set

- Tương tự như duyệt list, tuple

```
set_fruits = {'orange', 'lemon', 'apple', 'strawberry'}  
for fruit in set_fruits:  
    print(fruit)
```

apple
lemon
strawberry
orange

4. Set

□ Chiều dài, max, min, sum của set

- Cú pháp:

`len(ten_set)`

`max(ten_set)`

`min(ten_set)`

`sum(ten_set)`

```
set_nums = {1, 7, 2, 8, 3, 6}
print('Chiều dài:', len(set_nums))
print('Phần tử lớn nhất:', max(set_nums))
print('Phần tử nhỏ nhất:', min(set_nums))
print('Tổng các phần tử:', sum(set_nums))
```

4. Set

□ Tạo bản sao của set

- Cú pháp: `ten_set_dich = ten_set_nguon.copy()`

```
set_animals = {'bear', 'dog', 'lion', 'elephant', 'horse'}
print('set_animals:', set_animals)
set_animals_copy = set_animals.copy()
print('set_animals_copy:', set_animals_copy)
```

4. Set

□ Sắp xếp các phần tử trong set

- Cú pháp:

- Tăng dần: `sorted(ten_set)`

- Giảm dần: `sorted(ten_set, reverse=True)`

```
set_animals = {'bear', 'dog', 'lion', 'elephant', 'horse'}
print('Tăng dần:', sorted(set_animals))
print('Giảm dần:', sorted(set_animals, reverse=True))
```

Tăng dần: ['bear', 'dog', 'elephant', 'horse', 'lion']

Giảm dần: ['lion', 'horse', 'elephant', 'dog', 'bear']

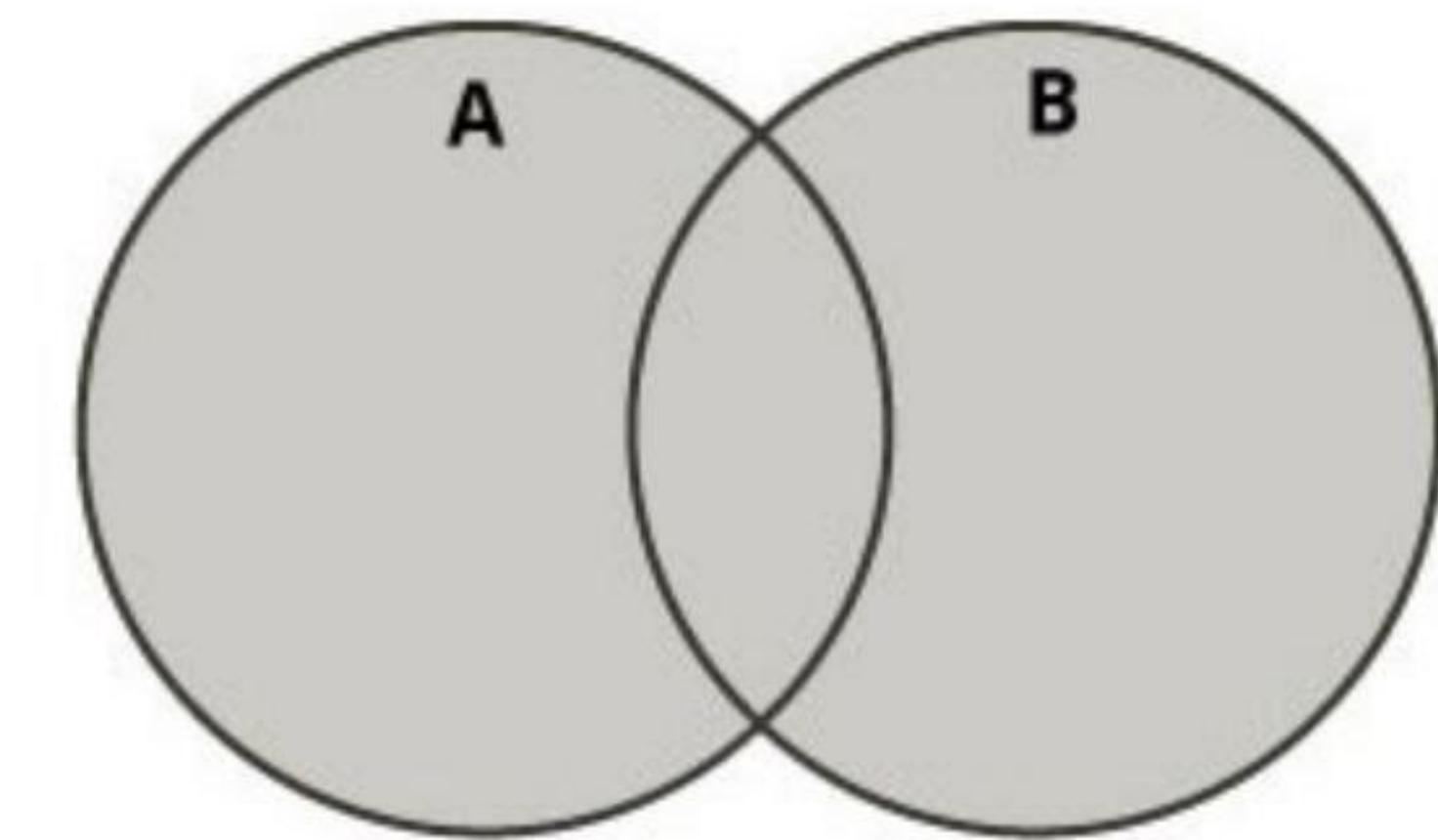
4. Set

❑ Các toán tử và phương thức trên set

- Set Union (phép hợp): Trả về tất cả các phần không trùng nhau của các set.
- Cú pháp: `set_union = set_1 | set_2` hoặc phương thức `union()`

```
set_1 = {1, 2, 4, 6, 8}  
set_2 = {1, 2, 3, 5, 7}
```

```
set_union = set_1 | set_2  
print(set_union)          # {1, 2, 3, 4, 5, 6, 7, 8}  
  
print(set_1.union(set_2)) # {1, 2, 3, 4, 5, 6, 7, 8}
```

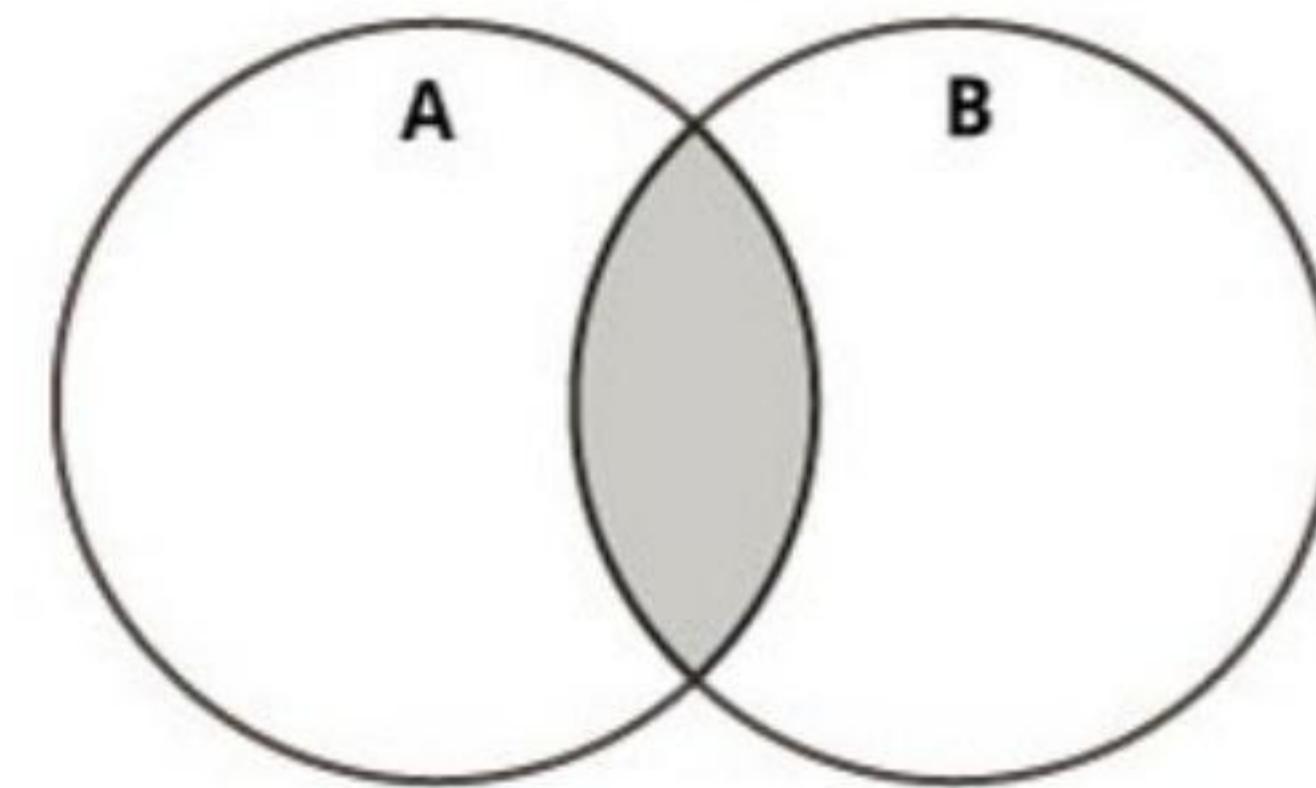


4. Set

❑ Các toán tử và phương thức trên set

- Set Intersection (phép giao): Trả về set mới các phần tử cùng xuất hiện (phần giao) trong các set.
- Cú pháp: `set_union = set_1 & set_2` hoặc phương thức `intersection()`

```
set_1 = {1, 2, 4, 6, 8}  
set_2 = {1, 2, 3, 5, 7}  
  
set_intersection = set_1 & set_2  
print(set_intersection)      # {1, 2}  
  
print(set_1.intersection(set_2))    # {1, 2}
```

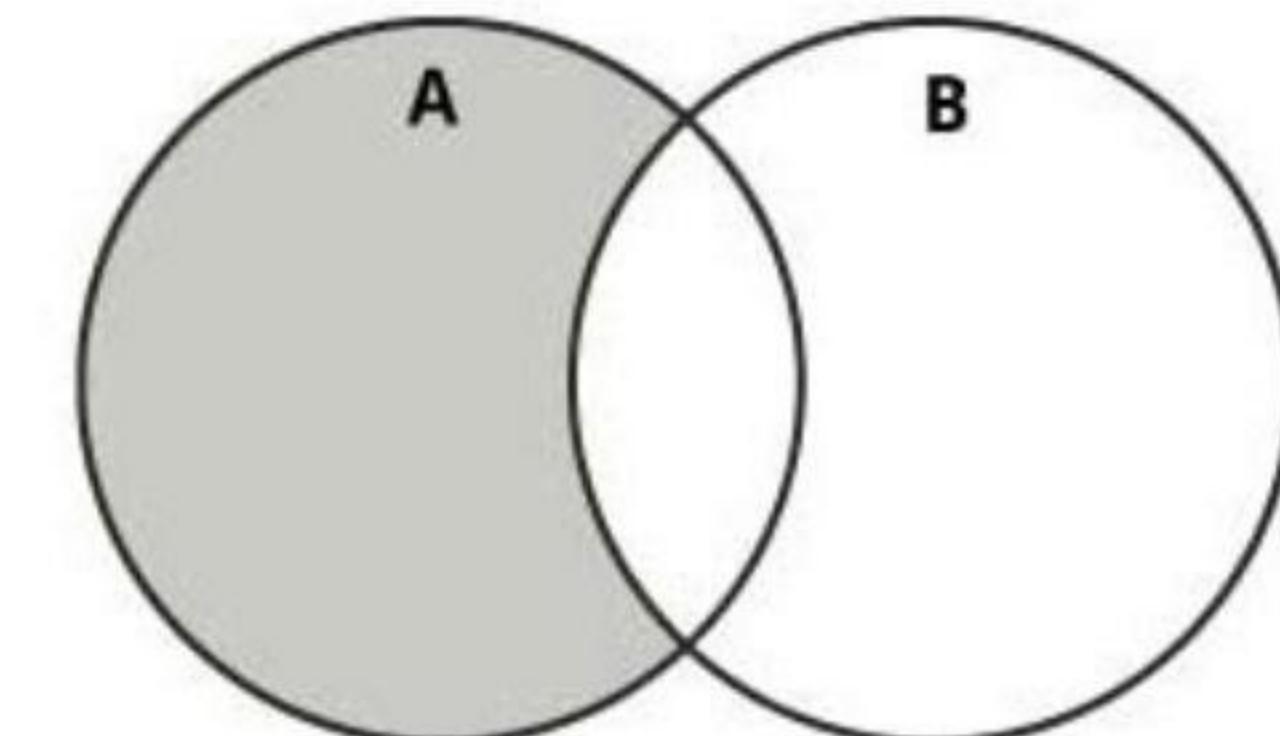


4. Set

❑ Các toán tử và phương thức trên set

- Set Difference (phép hiệu): Hiệu của A và B ($= A - B$) là tập hợp phần tử chỉ có trong A và không có trong B.
- Cú pháp: `set_difference = set_1 - set_2` hoặc phương thức `difference()`

```
set_1 = {1, 2, 4, 6, 8}  
set_2 = {1, 2, 3, 5, 7}  
  
set_difference = set_1 - set_2  
print(set_difference)      # {8, 4, 6}  
  
print(set_1.difference(set_2))  # {8, 4, 6}
```

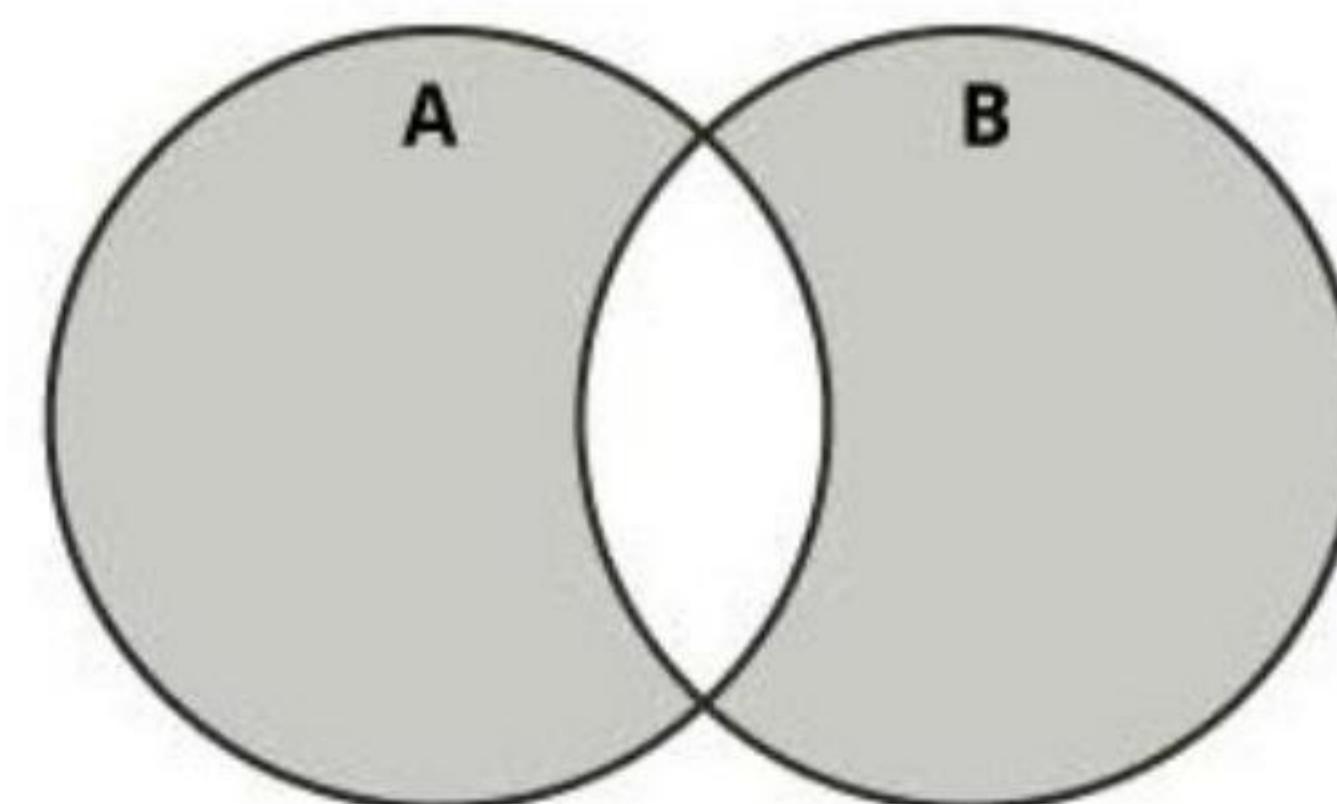


4. Set

❑ Các toán tử và phương thức trên set

- Set Symmetric Difference (phép bù): Bù của A và B là tập hợp những phần tử có trong A và B nhưng không phải phần tử chung của hai tập hợp này.
- Cú pháp: `set_symmetric_difference = set_1 ^ set_2` hoặc phương thức `symmetric_difference()`

```
set_1 = {1, 2, 4, 6, 8}  
set_2 = {1, 2, 3, 5, 7}  
  
set_symmetric_difference = set_1 ^ set_2  
print(set_symmetric_difference)      # {3, 4, 5, 6, 7, 8}  
  
print(set_1.symmetric_difference(set_2)) # {3, 4, 5, 6, 7, 8}
```





TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
TIT TRUNG TÂM TIN HỌC
TTH