# Kiểu `list`, `set` và `dictionary`

December 18, 2020

## 1 Giới thiệu

Trong bài học này chúng ta sẽ học về các kiểu dữ liệu `list`, `set` và `dictionary` trong python.

## 2 Kiểu dữ liệu `list`

### 2.1 Giới thiệu

`list` là kiểu dữ liệu cơ bản của python với cấu trúc :

```
[phần_tử_0, phần_tử_1, ..., phần_tử_n]
```

```
In [1]: # ví dụ
        a = [1, 2, 3]

        type(a)
```

```
Out[1]: list
```

Không có ràng buộc nào cho các phần tử trong list, nghĩa là : - Các phần tử có thể giống nhau. - Các phần tử có thể có kiểu dữ liệu khác nhau.

```
In [0]: # không có ràng buộc nào cho các phần tử trong list
        b = [1, 'a', ['c', 'd'], 1]

        b
```

```
Out[0]: [1, 'a', ['c', 'd'], 1]
```

### 2.2 Truy cập đến các phần tử trong list

Bạn có thể truy cập đến các phần tử trong `list` tương tự như cách truy cập đến các ký tự trong `str`.

```
In [0]: l = [0, 1, 2, 3, 4, 5, 6]

        # lấy phần tử đầu tiên
        print(l[0])
```

```
        # lấy phần tử cuối cùng
        print(l[-1])

        # lấy mỗi hai phần tử một lần, bắt đầu từ phần tử thứ 2
        print(l[1:-1:2])

        # lấy list thao thứ tự ngược lại
        print(l[::-1])
```

```
0
6
[1, 3, 5]
[6, 5, 4, 3, 2, 1, 0]
```

## 2.3 Một số thao tác với `list`

### 2.3.1 Thêm phần tử mới

Để thêm một phần tử mới, bạn có thể dùng toán tử cộng + hoặc dùng phương thức `.append()`.

```
In [4]: # ví dụ
        a = [1, 2, 3, 4]
        b = [5]
        print(a)
        a.append(b)
        print(a)
```

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

Lưu ý rằng `.append()` sẽ trực tiếp thêm vào `list`. Vì thế, hãy cẩn thận khi dùng `.append()`.

### 2.3.2 Thay đổi giá trị phần tử các giá trị trong `list`

Để thay đổi giá trị của phần tử trong `list`, bạn thực hiện phép gán như sau

```
In [7]: a = [1, 4, 9, 24]

        # thay đổi giá trị một phần tử
        a[-1] = 25
        print(a)
```

```
[1, 4, 9, 25]
```

```
In [8]: a = [1, 2, 3, 4]

        # thay đổi giá trị nhiều phần tử
        a[1:3] = ['a', 'b']
        print(a)
```

```
[1, 'a', 'b', 4]
```

**Lưu ý** : Khi thay đổi giá trị nhiều phần tử thì số lượng hai bên phải bằng nhau.

```
In [0]: a = [1, 2, 3, 4]
        a[1:3] = 'a'
        print(a)

[1, 'a', 4]
```

### 2.3.3 Xóa phần tử trong `list`

Để xóa một (hoặc nhiều) phần tử trong `list`, bạn có thể gán những phần tử đó bằng `list` rỗng `[]`.

```
In [9]: a = [1, 2, 3, 4]
        a[1:] = []
        print(a)

[1]
```

# 3 Kiểu dữ liệu `set`

## 3.1 Giới thiệu

`set` là một kiểu dữ liệu của python. Về cơ bản, `set` là một `list` nhưng các phần tử không trùng nhau. Nói cách khác, `set` là cách python dùng để thể hiện tập hợp toán học.

Để tạo một `set`, bạn có thể thực hiện theo cách sau

```
In [9]: # set sẽ được đánh dấu bằng cặp ngoặc nhọn {}
        s = {1, 2, 3, 4, 4, 1, 3}

        print(s)
        print(type(s))

{1, 2, 3, 4}
<class 'set'>
```

Ngoài ra, bạn có thể tạo ra `set` từ `list` bằng cách dùng hàm `set()`.

```
In [12]: s2 = set([1, 2, 'a', 'a'])
         print(s2)

{1, 2, 'a'}
```

### 3.2 Các thao tác trên set

Tuy có thể hiểu set là một list có các phần tử không trùng nhau nhưng bạn không thể truy cập đến các phần tử trong set giống như list.

```
In [13]: s2[1]
```

```
---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call last)

<ipython-input-13-8755b941c10d> in <module>
----> 1 s2[1]


TypeError: 'set' object does not support indexing
```

Một số các phép toán đối với set.

```
In [0]: a = set([1, 2, 3, 4])
        b = set([3, 4, 5, 6])

        print('Phép hợp hai tập hợp : ', a | b)
        print('Phép giao hai tập hợp : ', a & b)
        print('Phép lấy những phần tử thuộc a nhưng không thuộc b : ', a - b)
```

```
Phép hợp hai tập hợp :  {1, 2, 3, 4, 5, 6}
Phép giao hai tập hợp :  {3, 4}
Phép lấy những phần tử thuộc a nhưng không thuộc b :  {1, 2}
```

# 4 Kiểu dữ liệu dictionary

## 4.1 Giới thiệu

dictionary là một kiểu dữ liệu trong python với cấu trúc :

```
{khoá_1 : giá_trị_1, khoá_2 : giá_trị_2, ..., khoá_n : giá_trị_n}
```

Trong đó: - khoá_1, khoá_2, ..., khoá_n là không trùng nhau. - Các khoá có thể có kiểu str hoặc số nhưng không thể là list, set, ...

```
In [14]: # ví dụ dictionary
         d = {'a': 1, 'b': 2, 'c': 3}

         type(d)
```

```
Out[14]: dict
```

## 4.2 Truy cập đến một phần tử

Để truy cập đến một phần tử trong `dictionary`, bạn dùng cú pháp

```
<tên_dictionary>[<khoá_cần_lấy_giá_trị>]
```

```
In [18]: d['c']
```

```
Out[18]: 3
```

## 4.3 Một số thao tác trên `dictionary`

### 4.3.1 Thêm phần tử mới

Để thêm một phần tử mới vào `dictionary`, ta dùng cú pháp gán sau :

```
<tên_dictionary>[<khoá_mới>] = giá_trị
```

Lưu ý : `khoá_mới` không được phép trùng với các khoá đã có, nếu không sẽ chuyển thành thao tác thay đổi giá trị một phần tử.

```
In [20]: d['e'] = 5

         d
```

```
Out[20]: {'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5}
```

### 4.3.2 Thay đổi giá trị ứng với một khoá

Như lưu ý ở trên, bạn có thể dùng câu gán

```
<tên_dictionary>[<khoá_muốn_thay_đổi_giá_trị>] = giá_trị_mới
```

```
In [21]: d['d'] = 5

         d
```

```
Out[21]: {'a': 1, 'b': 2, 'c': 3, 'd': 5, 'e': 5}
```

### 4.3.3 Lấy ra các khoá

Để lấy ra các khoá bạn dùng phương thức `.keys()` như sau :

```
<tên_dictionary>.keys()
```

```
In [26]: k = list(d.keys())
         k
```

```
Out[26]: ['a', 'b', 'c', 'd', 'e']
```

Kết quả trả về của chúng ta là một đối tượng con của `list`. Để trả về `list`, bạn dùng thêm hàm `list()`.

```
In [28]: list(d.items())
```

```
Out[28]: [('a', 1), ('b', 2), ('c', 3), ('d', 5), ('e', 5)]
```

# 5 Từ khoá `in`

Từ khoá này cho phép bạn kiểm tra một giá trị có thuộc `list`, `set` cho trước hay không.

```
In [0]: # ví dụ dùng in với list
        l = [1, 2, 3, 4]

        print(1 in l)
        print(5 in l)
```

```
True
False
```

```
In [16]: # ví dụ dùng in với set
         s = {1, 2, 3, 4}

         print(1 in s)
         print(5 in s)
```

```
True
False
```

Còn đối với `dictionary`, từ khoá `in` sẽ kiểm tra giá trị đó có phải là một khoá của `dictionary` hay không.

```
In [0]: d = {'a' : 1, 'b' : 2, 'c' : 3}

        print('a' in d)
        print('e' in d)
```

```
True
False
```

Bạn còn có thể dùng `in` để kiểm tra một chuỗi có nằm trong một chuỗi khác hay không

```
In [30]: st1 = 'Hello'
         st2 = 'hello'
         st3 = 'Hello World!'

         print(st1 in st3)
         print(st2 in st3)
         print(st3.find(st1))
         print(st3.find(st2))
```

```
True
False
0
-1
```

Ngoài ra, từ khoá `in` còn cho bạn khả năng lặp qua các phần tử `list`, `set` và `dictionary` như sẽ trình bày ở phần sau.

# 6 Vòng lặp for và các kiểu dữ liệu tập hợp

Đối với một số ngôn ngữ lập trình, để lặp qua các phần tử trong một `list`, bạn cần phải dùng thông qua `index` của từng phần tử. Chẳng hạn :

```
In [17]: l = ['a', 'b', 'c', 'd']

         for i in range(len(l)) :
             print(l[i])
```

```
a
b
c
d
```

Tuy nhiên, với python, bạn có thể lặp trực tiếp qua từng phần tử của `list` như sau :

```
In [22]: l2 = ['e', 'f', 'g', 'h']

         for e in l2 :
             print(e)

         list(range(10))
```

```
e
f
g
h
```

```
Out[22]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Đối với `dictionary`, người ta thường lặp qua cặp khoá - giá trị như sau :

```
In [0]: d1 = {'a' : 1, 'b' : 2}

        for k, v in d1.items() :
            print(k, v)
```

```
a 1
b 2
```

Ngoài ra, bạn có thể dùng `for ... in ...` để lặp qua từng ký tự của một chuỗi như sau :

```
In [0]: s1 = 'Hello'
        for e in s1 :
            print(e)
```

```
H
e
l
l
o
```

# 7 Tạo list theo cách python

**Bài tập** : Tạo ra một `list` chứa bình phương của các số từ 1 đến 9?
Thông thường, bạn có thể sẽ viết như sau :

```
In [23]: li = []
         for i in range(1, 10) :
             li = li + [i * i]

         li

Out[23]: [1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Tuy nhiên, trong python, bạn có thể viêt gọn lại như sau :

```
In [25]: li = [i * i for i in range(1, 10)]
         li

Out[25]: [1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Bạn có thể hiểu như sau `li` là danh sách của các giá trị `i * i` với `i` nằm trong `range(1, 10)`.